

ARBEITSBERICHTE DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

HERAUSGEBER: H. Billing, W. Händler, U. Herzog,
F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees, H. Niemann,
D. Seitzer, B. Schmidt, H. J. Schneider, H. Wedekind

FUNKTIONALER ENTWURF UND AUTOMATISCHE
SYNTHESE HOCHINTEGRIERTER SCHALTkreISE

Wolfgang Brendel

DISSERTATION

BAND 17 · NUMMER 4 · ERLANGEN · AUGUST 1984

1984 № 4 18



ISSN 0344 - 3515



Digitized by the Internet Archive
in 2026

FUNKTIONALER ENTWURF UND AUTOMATISCHE SYNTHESE HOCHINTEGRIERTER SCHALTKREISE

Der Technischen Fakultät
der Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR - INGENIEUR

vorgelegt

von

Wolfgang Brendel

Erlangen 1984

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung: 13. Feb. 1984

Tag der Promotion: 22. Juni 1984

Dekan: Prof. Dipl.-Ing. G. Vetter

Berichterstatter: Prof. Dr. U. Herzog

Prof. Dr. H.J. Schneider

KURZFASSUNG

Ausgehend vom Beispiel eines hochintegrierten Schaltkreises zum Sortieren von Elementen nach dem Prinzip 'BUBBLE SORT' werden Eigenschaften von Rechner-Entwurfssprachen zur funktionalen Spezifikation digitaler, synchroner Schaltwerke entwickelt. Diese sind Grundlage für das Konzept eines Synthese-Systems, das es gestattet sowohl die Topologie, als auch schaltungstechnische Details aus der funktionalen Beschreibung abzuleiten.

Als formales Hilfsmittel zur Beschreibung der syntaktischen und technologischen Strukturen werden Graph-Grammatiken, zur Formalisierung des Syntheseprozesses die Syntaxanalyse auf Graphen verwendet. Dies ermöglicht sowohl die Erfassung logischer und struktureller Eigenschaften des Entwurfs, als auch die Beschreibung zeitlicher Randbedingungen.

Die Validierung der funktionalen Spezifikation des Entwurfs erfolgt durch Simulation auf der Architekturebene. Da algorithmischer Ablauf und zeitliches Zusammenwirken der einzelnen Komponenten im Vordergrund der Betrachtung stehen, ist die Simulation umfangreicher und komplexer Entwürfe möglich, die in ihrer Gesamtheit auf technologieorientierten Beschreibungsebenen aus Aufwandsgründen nicht mehr untersucht werden können. Die Erweiterung der Entwurfssprache zur Modellierung spezieller Konstruktionen hochintegrierter Schaltkreise macht jedoch die Verwendung einer fünfwertigen Logik bei der Simulation notwendig.

Die beim Entwurf des im Rahmen dieser Arbeit in NMOS-Technologie entwickelten und gefertigten 'BUBBLE SORT' - Bausteins gewählte Vorgehensweise mit einer funktionalen Beschreibung auf der Register-Transferebene zu beginnen und den Entwurf durch Simulation zu validieren sowie seine Leistung zu verbessern zeigt, daß eine systematische Entwicklung, beginnend auf einem hohen Abstraktionsniveau, mit relativ geringem Aufwand und den bereits verfügbaren Mitteln zu greifbaren Ergebnissen führen kann.

INHALTSVERZEICHNIS

Einleitung	1
1. Entwurfs- und Synthese-Systeme	4
1.1. Entwurfsmethodik für hochintegrierte Schaltkreise	4
1.1.1. Entwurfsprozeß und Beschreibungsebenen	5
1.1.2. Prinzipien und Ziele der Entwurfsmethodik	9
1.1.3. Zellkonzept und Zellbibliothek	14
1.2. Struktur rechnergestützter Entwurfssysteme	23
1.2.1. Systeme und Programme	23
1.2.2. Entwurfssprachen	25
1.2.3. Funktionale Simulation	27
1.2.4. Layout	29
1.2.5. Entwurfsregeln und topologische Analyse	32
1.2.6. Elektrische Simulation	39
1.2.7. Generierung von Testmustern	40
1.3. Silicon Compiler	43
1.3.1. CERES Synthese System	43
1.3.2. Vergleich verschiedener Synthese-Systeme	48
2. VLSI - Fallstudie BUBBLE-SORT Baustein	53
2.1. Sortieralgorithmen für VLSI	53
2.2. Funktionale Spezifikation und Simulation	55
2.2.1. ERES Beschreibung des SORT Bausteins	56
2.2.2. Simulation und Leistungsbewertung	62
2.3. Entwurf elementarer Zellen	65
2.3.1. Register und Vergleicher	65
2.3.2. Lokale Steuerlogik	69
2.3.3. Globale Steuerlogik	74
2.3.4. Layout des SORT Bausteins	74
2.4. Elektrische Simulation	77
2.4.1. Physikalische Parameter für SPICE	77
2.4.2. Register und Vergleicher	79
2.4.3. Lokale und globale Steuerlogik	84
2.4.4. Vergleich elektrische und funktionale Simulation	89
2.5. Kritik der funktionalen Spezifikation	91
3. Attributierte Graph-Grammatiken	94
3.1. Quasihierarchische Graphen	95
3.2. Ersetzungsregeln auf Graphen	98
3.3. Ableitung und programmierte Graph-Grammatik	102
3.4. Syntaxanalyse und Implementierungsaspekte	106

4.	Spezifikation und Synthese des Datenpfades	109
4.1.	Verknüpfung logischer Signale	109
4.1.1.	Elementare Operatoren	111
4.1.2.	Logikelemente für VLSI	117
4.2.	Register und Speicher	125
4.2.1.	Register und Latch	125
4.2.2.	Adressierbare Speicher	129
4.3.	Reguläre Strukturen	130
4.3.1.	Wortstruktur und Busse	130
4.3.2.	Replikationsoperator	131
4.3.3.	Reguläre Logikfelder	131
5.	Definition und Simulation des Zeitmodells	145
5.1.	Spezifikation von Takten	145
5.1.1.	Takte und Mehrphasenuhr	146
5.1.2.	Abgeleitete Taktsignale	146
5.2.	Verzögerungszeiten	149
5.2.1.	Zeitverhalten von Register und Latch	150
5.2.2.	Terminals	153
5.3.	Zeitbeschreibung der Zellbibliothek	156
5.3.1.	Attribute für logische Verknüpfungen	156
5.3.2.	Attribute für PLA-Logik	158
6.	Nichtprozedurale Kontrollstrukturen für die Steuerlogik	162
6.1.	Uhrgesteuerte Technik	163
6.1.1.	Parallele Abläufe	163
6.1.2.	Sequentialisierung	168
6.2.	Deterministische Automaten	168
6.2.1.	Funktionale Spezifikation	169
6.2.2.	Beispiel Ampelsteuerung	170
6.2.3.	Ermittlung der Zustandsüberföhrungsfunktion	174
7.	Zusammenfassung der Ergebnisse und Ausblick	177
7.1.	CERES, Rechner-Entwurfssprache für VLSI	179
7.2.	Testfreundlicher Entwurf	180
7.3.	Entwurf, Simulation und Messungen	182
7.4.	Synthese, Platzierung und Verdrahtung	183
	Schlußbemerkung	184
	Literaturverzeichnis	185
	Stichwortverzeichnis	194
	Anhang: CERES Beschreibung des BUBBLE-SORT Bausteins	197

EINLEITUNG

Durch die Technik der Größtintegration und die damit mögliche Realisierung komplexer Systeme auf einem Chip werden neue Anforderungen an den Entwurfsprozeß und die Fertigung digitaler Systeme gestellt. Die bisherige lose Kopplung der einzelnen Entwurfsphasen hatte eine Trennung der verschiedenen Tätigkeiten vom Systementwurf bis zur Schaltkreisentwicklung und dessen technologischer Realisierung zur Folge. Bei der Entwicklung hochintegrierter Schaltungen sind dagegen die einzelnen Entwicklungsschritte stark voneinander abhängig, da bereits die Architektur eines Systems von den Eigenschaften seiner technologischen Realisierung geprägt wird. Bereits während des Entwurfs sind Aussagen über die zu erwartende Leistung des Systems notwendig.

Der zunehmende Bedarf an hochintegrierten Bausteinen und die durch Spezialisierung geringere Anwendungsbreite der Schaltkreise macht die Automatisierung des Entwurfsprozesses unumgänglich. Das Ziel einer Entwurfsautomatisierung ist es, dem Entwickler ein Werkzeug zur Verfügung zu stellen, das eine Verkürzung der Entwurfszeit und durch automatische Prüfungen eine Qualitätsverbesserung ermöglicht. Die vorliegende Arbeit ergänzt deshalb die Sprachkonstrukte der konventionellen Rechner-Entwurfssprache ERES um Operatoren und Datentypen zur Beschreibung hochintegrierter Schaltkreise.

Rechner-Entwurfssprachen erlauben eine funktionale Beschreibung des zu entwickelnden Systems und sind deshalb als Ausgangsbasis für eine automatische Synthese digitaler Schaltungen hervorragend geeignet. Die Validierung eines Entwurfs kann durch Simulation der Operationen und des zeitlichen Ablaufgeschehens durchgeführt werden, ohne daß der gesamte Entwurfsprozeß durchlaufen werden muß. Ausgehend von der funktionalen Beschreibung soll dann mit Hilfe einer automatischen Synthese eine Herstellungsvorschrift für die Realisierung des Systems als hochintegrierter Baustein gewonnen werden.

Um die Technik der automatischen Synthese hochintegrierter Schaltkreise einer formalen Behandlung zugänglich zu machen, ist eine umfassende Modellbildung notwendig. Als Modell dient die abstrakte Beschreibung der topologischen, funktionalen und zeitlichen Struktur des Entwurfs durch

einen attributierten Graphen. Dadurch werden alle für die Synthese wesentlichen Eigenschaften repräsentiert, Details spezieller technologischer Prozesse jedoch unterdrückt. Die syntaktische Struktur und kontextabhängige Eigenschaften der verwendeten Rechner-Entwurfssprache werden durch Graph-Grammatiken definiert.

Die technologische Realisierung eines Entwurfs besteht aus verschiedenen Halbleitermaterialien, die in mehreren Schichten übereinander angeordnet werden. Jede dieser Schichten wird mit Hilfe von Masken so gestaltet, daß Transistoren, Gatter, Register, Speicher und komplexere Zellen entstehen. Formalisiert man die Eigenschaften dieser Zellen ebenfalls als Graph, können die zulässigen Kombinationen wiederum durch eine Graph-Grammatik definiert werden.

Das Problem der automatischen Synthese hochintegrierter Schaltungen aus einer funktionalen Beschreibung kann dann mit Hilfe der Syntaxanalyse auf Graph-Grammatiken gelöst werden. Es ist dabei zu analysieren, ob der vorliegende Entwurf, repräsentiert durch einen Graphen, in der Grammatik der verwendeten Zellen ableitbar ist. Der aus der Syntaxanalyse hervorgehende Syntaxgraph liefert dabei eine Vorschrift zur Synthese des Systems, da er die exakte Kombination der Zellen enthält, die eine Realisierung des Entwurfs darstellen. Die der automatischen Synthese zugrunde liegende Methode geht also von der funktionalen Beschreibung des Entwurfs aus und analysiert, ob dieser durch eine zulässige Kombination elementarer Einheiten der Technologie synthetisiert werden kann.

Diese Vorgehensweise ist vergleichbar mit der Klassifizierung von Pflanzen in der Biologie. Beschreibt man die Zellen einer Pflanzenart als Knoten und die Nachbarschaften zwischen ihnen durch Kanten eines Graphen, kann der Wachstumsprozeß durch eine Graph-Grammatik formalisiert werden. Die Menge aller Graphen, die mit dieser Grammatik erzeugt werden, beschreibt dann alle Pflanzen der Art, in verschiedenen Stadien des Wachstums. Zur Klassifizierung einer Pflanze als zu der Art gehörend genügt es nachzuweisen, daß die Zellstruktur in der Grammatik ableitbar ist. Auf die Synthese hochintegrierter Schaltkreise übertragen bedeutet dies, daß der Entwurf mit der gewählten Zellbibliothek realisierbar ist.

Nach einer Einführung in die Entwurfsmethodik für hochintegrierte Schaltkreise und einer Darstellung der Struktur des rechnergestützten

Entwurfs- und Synthesesystems CERES wird im zweiten Kapitel ein VLSI-Baustein zum Sortieren von Daten entwickelt. Anhand dieser Fallstudie werden die wichtigsten Methoden für den Entwurf hochintegrierter Schaltkreise und die dabei verwendeten Werkzeuge dargestellt. Sie bilden die Ausgangsbasis für die Definition einer Entwurfssprache zur funktionalen Spezifikation und Simulation. Die theoretischen Grundlagen für die Formalisierung des Syntheseprozesses durch attributierte Graph-Grammatiken enthält das dritte Kapitel. Diese werden dann in den folgenden Kapiteln auf die Beschreibung und die Synthese von Datenpfad und Steuerlogik sowie der Uhr angewendet.

Das abschließende Kapitel faßt die wesentlichen Ergebnisse dieser Arbeit zusammen und gibt Anregungen für weitere Forschungen und Entwicklungen.

*We always got to develop
tomorrows technique with
yesterdays equipment.*

Tom Williams

1. ENTWURFS- UND SYNTHESE-SYSTEME

Der Rechner als Werkzeug zur Lösung komplexer Aufgaben ist wesentlicher Bestandteil unserer technologisch ausgerichteten Welt geworden. Die Erfindung hochintegrierter Schaltkreise, mit Tausenden von Transistoren auf wenigen Quadratmillimetern Silizium, hat das Erscheinungsbild des Rechners in den letzten Jahren entscheidend geändert. Nicht länger ist der Betrieb von Rechnern verbunden mit den klimatisierten Räumen eines Rechenzentrums, vielmehr werden heute Mikroprozessoren in großer Anzahl im Büro und im Fertigungsbereich eingesetzt. Die rasch fortschreitende Entwicklung der Mikroelektronik hat jedoch dazu geführt, daß hochintegrierte Schaltkreise neben ihrer Anwendung in Rechnern auch bei Konsumgütern wie Uhren, Kraftfahrzeugen, Taschenrechnern, Unterhaltungselektronik und sogar Spielzeug eingesetzt werden. Die Anwendung von Mikroprozessoren in Waschmaschinen und anderen Haushaltsgeräten ist dafür ein bekanntes Beispiel.

Die Reduktion der Transistorgeometrie hat im letzten Jahrzehnt zu einer jährlichen Verdopplung der Schaltungskomplexität, gemessen an der Zahl der Transistoren, geführt. Trotz der weiter fortschreitenden Technologie hat sich seit Beginn der achtziger Jahre diese Entwicklung verlangsamt, da mit den gebräuchlichen Entwurfshilfsmitteln Schaltkreise mit mehr als 100.000 Transistoren weder entworfen noch simuliert und getestet werden konnten. Die Nutzung der Höchstintegrationstechnik erfordert deshalb die Entwicklung neuer Entwurfsmethoden und geeigneter Werkzeuge.

1.1. ENTWURFSMETHODIK FÜR HOCHINTEGRIERTE SCHALTKREISE

Die Größe der graphischen Darstellung des Layouts einer integrierten Schaltung gibt ein anschauliches Beispiel der zunehmenden Integrationsdichte. Ein Papier im Format DIN A 0 reichte 1970 für den Entwurf eines Chips mit 5mm Kantenlänge, bei einer gezeichneten Leiterbahnbreite von

3mm. Eine Zeichnung gleicher Dichte hätte 1980 bereits 10 qm für die Darstellung eines gleich großen Chips benötigt. Ende der achtziger Jahre würde eine solche Zeichnung bereits ein Fußballfeld bedecken (LOSLEBEN 82). Manuelle Methoden für Entwurf und Layout reichen deshalb nicht mehr aus, rechnergestützte Entwurfsmethoden sind wesentliche Voraussetzung zur Beherrschung der Komplexität.

1.1.1. Entwurfsprozeß und Beschreibungsebenen

Die, bei der Entwicklung eines digitalen Schaltwerkes, bisher übliche Trennung von System-, Schaltkreis- und technologischer Entwicklung ist beim Einsatz der Höchstintegrationstechnik nicht mehr möglich, da Entwurfsentscheidungen die auf einem dieser Gebiete getroffen werden, die anderen beeinflussen. So wird die globale Struktur eines hochintegrierten Rechenwerkes und seine Aufteilung in Moduln auch durch die Begrenzung der Anzahl möglicher Anschlußpunkte bestimmt. Während die VLSI-Technik ('Very Large Scale Integration') ein rasches Anwachsen der in einem Baustein realisierbaren Funktionen mit sich gebracht hat, erhöhte sich die Anzahl der externen Anschlüsse nur unwesentlich.

Diese gegenseitige Beeinflussung von Aspekten der Architektur und Implementierung (BLAAUW 72) macht es notwendig, bei der Spezifikation eines Entwurfs, ausgehend von einer abstrakten, funktionalen Beschreibung, diese schrittweise zu verfeinern. Der Entwurf wird dabei auf verschiedenen Beschreibungsebenen spezifiziert und simuliert. Dabei auftretende Fehler erzwingen die Wiederholung eines oder mehrerer Schritte des Entwurfs (LIPP 82). Diese verschiedenen Ebenen unterscheiden sich durch die verwendeten Methoden, Werkzeuge und den Grad der Abstraktion, mit der Struktur und Verhalten des Entwurfs beschrieben werden. Die in Tabelle 1.1 dargestellte Aufteilung der Beschreibungsebenen orientiert sich an den Bedürfnissen des Entwicklungsprozesses hochintegrierter Schaltkreise und den dafür eingesetzten Entwurfswerkzeugen.

Systemebene

Auf der Systemebene wird der Entwurf als ein Netz von synchronen Teilsystemen beschrieben, die im allgemeinen asynchron gekoppelt sind.

Ebene	Beschreibungsmittel	abstraktes Modell	Beschreibungsinhalte
System	Systementwurfssprachen	Petri-Netze Warteschlangenmodelle	Kommunikation und Synchronisation der asynchron gekoppelten Teilsysteme
Register Transfer	Rechnerentwurfssprachen	Automaten Register-Transfer	Synchrone Systeme mit globaler Zeitbeschreibung
Gatter	Schaltplan	Gatter-Netzwerke Wahrheitstafeln	logische Verknüpfungen diskrete Verzögerungszeiten
Transistor	Schaltnetz von Bauelementen	Funktionen von Spannung / Strom	elektrisches Verhalten Zeitkontinuum
Layout	geometrische Darstellungen	Kombination von Polygonzügen	Anordnung und Dimension technologischer Elemente
Technologie	Differentialgleichungen	physikalische Modelle	Verteilung von Ladungsträgern

Tabelle 1.1: Beschreibungsebenen und Inhalte

Die Beschreibungsinhalte sind die Mechanismen zur Synchronisation der Teilsysteme und ihre gegenseitige Abhängigkeit. Wesentliches Ergebnis einer Analyse des Gesamtsystems sind Leistung und Lebendigkeit des Systems bei einer angenommenen Leistungsfähigkeit der Komponenten. Diese werden nur in ihrem Verhalten in Bezug auf ihre Umgebung, nicht jedoch in ihrem inneren Ablauf beschrieben. Eine Bewertung des Entwurfs auf der Systemebene kann entweder analytisch (Warteschlangenmodelle, Petri-Netze) oder durch Simulation vorgenommen werden.

Register-Transfer Ebene

Die Register-Transfer Ebene beschreibt synchrone Teilsysteme in ihrer Architektur durch Spezifikation des Befehlssatzes und seiner Semantik. Das Teilsystem wird dabei sowohl in seiner Struktur als auch in seinem algorithmischen Verhalten beschrieben. Als Beschreibungsmittel dienen Rechnerentwurfssprachen, deren Elemente direkten Bezug zu üblichen Hardwarekomponenten aufweisen. Dabei werden sowohl die verwendeten Elemente als auch die Datenpfade und Operationen spezifiziert. Die Beschreibung der Architektur kann modular aufgebaut sein, so daß die Semantik eines Befehls auf die Wirkungsweise mehrerer Mikrobefehle zurückgeführt wird. Rechnerentwurfssprachen erlauben damit die Entwicklung und Simulation von Hardware-Aktionen und Mikroprogrammen. Bereits auf der Register-Transferebene ist eine genaue Spezifikation des zeitlichen Verhaltens der einzelnen Komponenten eines Entwurfs notwendig. Die Simulation erlaubt dabei den Nachweis der Leistungsfähigkeit einer Architektur bezüglich der Effizienz des Befehlssatzes und seiner Implementierung. Die Ergebnisse der Simulation dienen auf der Systemebene zur Validierung der dort für das Teilsystem vorausgesetzten Leistungsmerkmale.

Gatterebene

Die Gatterebene beschreibt die Implementierung der Architektur durch ein Netz von logischen Gattern, Registern und speichernden Elementen. Die auf der Register-Transfer Ebene vorhandene Information über die Semantik einer logischen Verknüpfung bei der Ausführung eines Befehls geht auf der Gatterebene im allgemeinen verloren. Algorithmische Abläufe sind in der Struktur von Gattern nicht mehr zu erkennen. Die Spezifikation des Entwurfs auf Gatterebene ist bei konventioneller Realisierung mit Standard-Bausteinen Vorlage für die Fertigung. Für die Höchstintegrationstechnik ist eine Darstellung auf Gatterebene nur bedingt geeignet, da Implementierungsmethoden für VLSI-Schaltkreise teilweise die Verwendung anderer Strukturen wie PLA's ('Programmable Logic Array', siehe Abschnitt 4.3.3) und ROM ('Read Only Memory') nahelegen, deren interne Funktion (Belegung mit Transistoren) im Schaltplan nicht enthalten ist. Die Simulation eines Entwurfs auf der Gatterebene ermöglicht detaillierte Aussagen über das logische und zeitliche Verhalten eines Ent-

wurfes und eventuelle Störungen durch Signallaufzeiten bei gleichzeitigem Schalten mehrerer Eingänge eines Gatters.

Transistorebene

Die logische Funktion eines Gatters erhält auf der Transistorebene eine technologieabhängige Interpretation. Die Realisierung einer Funktion kann dabei selbst bei gleicher Technologie durch verschiedene schaltungstechnische Varianten vorgenommen werden. Die Darstellung des Schaltkreises auf der Transistorebene durch ein Netz von Bauelementen ist die Grundlage einer elektrischen Simulation. Dabei werden bezüglich der Schaltkreisparameter wie Transistor-Dimensionierung Annahmen getroffen, die mittels einer Simulation des elektrischen und zeitlichen Verhaltens überprüft werden. Mehrfache Modifikationen mit anschließender Simulation führen schließlich zu einem Parametersatz, der als Grundlage für den Entwurf des Layouts dient (vergleiche Abschnitt 2.4).

Layout-Ebene

Die geometrische Anordnung von Komponenten, die Dimensionierung der Transistoren und Leitungen werden durch die Layout-Beschreibung festgelegt. Die Konstruktion eines Gatters aus verschiedenen Transistoren hängt von der verwendeten Technologie ab (siehe MUROGA 82). Durch die Layoutbeschreibung werden Masken für die einzelnen Prozeß-Schritte definiert, die eine bestimmte Dotierung oder Metallisierungslage repräsentieren. Die Eingabe des Layouts erfolgt mit Grafik-Editoren, welche eine interaktive Arbeitsweise unterstützen. Die rechnerinterne Darstellung der geometrischen Daten dient zur Erzeugung von Maskenbändern, die bei der Herstellung des Schaltkreises die Schnittstelle zum technologischen Prozeß bilden (MEAD 80). Eine Simulation des Schaltungsverhaltens auf der Layoutebene ist nur nach einer Rückgewinnung der Schaltungsinformation durch Schaltungs-Extraktoren (circuit extractor) möglich, die aus den geometrischen Flächen Netze von Transistoren erzeugen. Diese dienen in einer maschinenlesbaren Form als Eingabe für Programme zur Schaltkreis-Simulation. Ein Vergleich dieser Simulationsergebnisse mit denen einer höheren Beschreibungsebene ermöglicht die Überprüfung der korrekten Implementierung der Schaltung auf Layout-Ebene.

Technologische Ebene

Die technologische Ebene ist für den Entwickler einer integrierten Schaltung durch die die Layout-Erstellung und die elektrische Simulation beeinflussenden Eigenschaften des verwendeten Prozesses von Bedeutung. Dies sind die Art und Anzahl der Masken, welche für den herstellerepezifischen Prozeß verwendet werden und physikalische Parameter, die für eine elektrische Simulation notwendig sind. Eine Simulation des Prozesses zur Bestimmung der zu erwartenden Eigenschaften von Transistoren basiert auf physikalischen Modellen und dient zur Optimierung laufender beziehungsweise zur Entwicklung neuer Prozesstechnologien.

Beim Entwurf eines hochintegrierten Schaltkreises werden alle der obigen Beschreibungsebenen in verschiedenen Phasen des Entwurfsprozesses durchlaufen. Die angegebenen Ebenen lassen sich jedoch abhängig von der angewandten Entwurfsmethodik und den dabei eingesetzten Hilfsmitteln weiter verfeinern. Grundlegend für die Anwendbarkeit einer Entwurfsmethode ist jedoch weniger die Differenzierung der Ebenen, als die Integration der ihnen zugeordneten rechnergestützten Werkzeuge zu einem Entwurfssystem.

1.1.2. Prinzipien und Ziele der Entwurfsmethodik

Die Vorgehensweise beim Entwurf prägt die Struktur und Funktion eines rechnergestützten Entwurfssystems. Für das Verständnis eines solchen Entwurfssystems ist es deshalb notwendig, die Entwurfsmethodik und die mit ihr verfolgten Ziele zu beschreiben. Die hier skizzierte Entwurfsmethode soll Rechnerarchitekten mit Erfahrung auf dem Gebiet des Entwurfs, aber begrenztem Wissen über die Physik der Bauelemente hochintegrierter Schaltkreise in die Lage versetzen, die architektonischen Möglichkeiten der VLSI-Technik zu nutzen, ohne die bei der Verwendung konservativer Techniken unvermeidlichen Leistungseinbußen hinnehmen zu müssen.

Charakteristik der Entwurfsmethode

Die hier eingeführte Entwurfsmethode unterscheidet sich von konventionellen Methoden des Hardware-Entwurfs durch die Vereinigung wesentlicher Merkmale der zergliedernden ('top down') und der aufbauenden ('bottom up') Methoden. Besondere Bedeutung erlangt die beschriebene Vorgehensweise durch die Möglichkeit der rechnergestützten Synthese von Schaltkreisen aus vordefinierten Zellen (vergl. Abschnitt 1.3.1). Das Zusammenwirken der beiden Entwurfsstrategien zeigt Bild 1.1. Das Synthese-System verbindet dabei beide Teile zu einem geschlossenen Entwurfssystem.

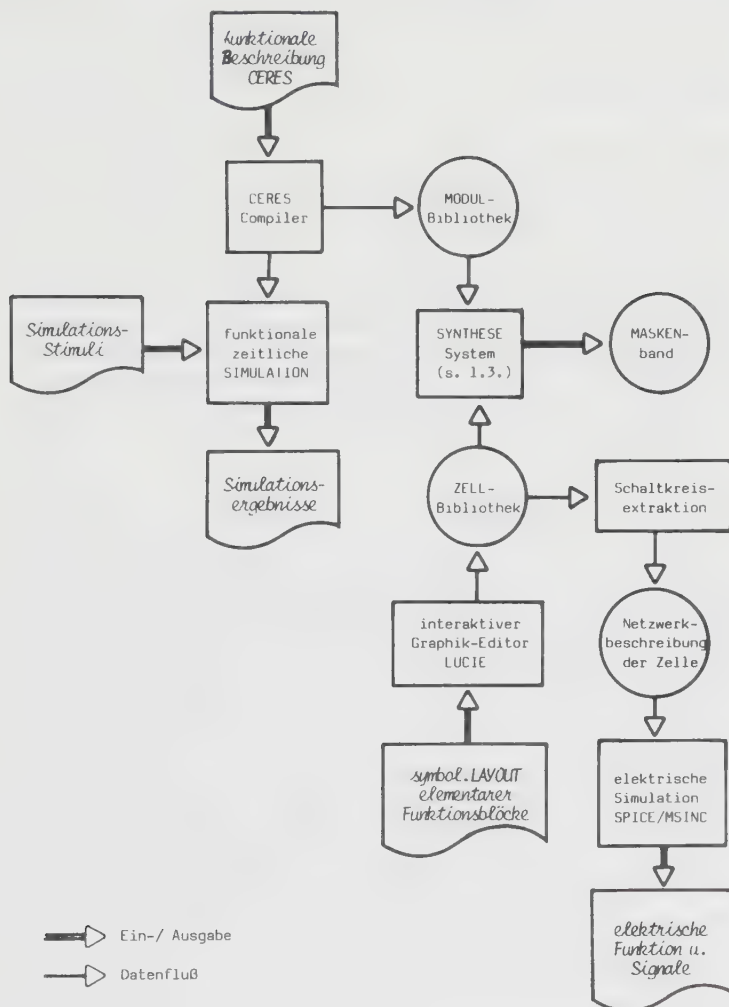
Funktionale Spezifikation 'top down'

Der funktionale Entwurf geht einher mit einer Modularisierung des Entwurfs und der Zergliederung in Teilsysteme beschränkter Komplexität unter Verwendung einer Rechner-Entwurfssprache. Bei einem hierarchischen Entwurf besteht dieser aus Teilsystemen, die über Schnittstellen untereinander kommunizieren. Die Schnittstellen des Gesamtsystems werden auf der obersten Modularisierungsebene spezifiziert. Die Topologie eines Systems ist damit die Struktur, die sich aus einer Reihe von geschachtelten Teilsystemen ergibt, die untereinander verbunden sind.

Für jeden dieser Moduln werden die Schnittstellen zu anderen Moduln spezifiziert, seine innere Struktur und Funktion beschrieben. Dabei werden insbesondere die verwendeten Teilsysteme und deren Verbindung, primitive Elemente wie Register, Speicher und logische Verknüpfungen sowie deren Datenpfade festgelegt. Die Deklarationen der primitiven Elemente enthalten bereits die zeitlichen Eigenschaften der Objekte und der Operationen. Damit sind auch die funktionalen und zeitlichen Eigenschaften der entsprechenden Teilsysteme festgelegt (vergleiche Kapitel 4 bis 6).

Die funktionale Beschreibung des Entwurfs mit Hilfe einer Rechner-Entwurfssprache ermöglicht bereits in der Spezifikationssphase die Simulation der Funktion eines Entwurfs. Dadurch ist eine Bewertung der Leistungsfähigkeit der Architektur und eine Optimierung des Befehlssatzes und anderer Charakteristika wie Datenpfadbreite und Speicherge-

TOP DOWN FUNKTIONSENTWURF



BOTTOM UP ZELLENTWURF

Bild 1.1.: Entwurfsmethodik für den Schaltkreisentwurf

schwindigkeit möglich, ohne daß spätere Phasen des Entwurfsprozesses dadurch einbezogen werden müßten.

Zellentwurf 'bottom up'

Für jede primitive Funktion (Register, Terminal) eines Moduls wird das Layout einer Zelle entworfen, welche die Logik durch eine Transistor-schaltung realisiert. Dabei werden Daten- und Kontrollfluß orthogonal angeordnet, um eine günstige, rasterartige Layout-Struktur zu erzeugen (vergl. MEAD 80). Es wird keine Reduktion der Transistoranzahl bei der Realisierung der Zellen angestrebt, da der Platzbedarf für Verbindungs-leitungen den der Transistoren überwiegt und diese teilweise unter den Metalleitungen plaziert werden können. Beim Entwurf der Steuerlogik und Taktsignale wird von den in der funktionalen Beschreibung spezifizierten Kontrollstrukturen und Aktivierungsbedingungen ausgegangen. Zur Realisierung werden stark reguläre Strukturen verwendet, um eine gute Änderungsfreundlichkeit zu erreichen.

Diese Zellen werden entsprechend der Topologie des Entwurfs so plaziert, daß benachbarte Zellen mit gemeinsamen Verbindungsleitungen direkt aneinander stoßen (wiring by abutment). Dazu kann es notwendig sein, die Zellen zu strecken, um eine Anpassung der Steuerlogik an das entstandene Funktionsfeld zu erreichen. Im letzten Schritt werden dann die Ein- und Ausgabe-Schaltungen für die Verbindung zu anderen Schaltkreisen eingesetzt und die Steuer- und Datenleitungen mit den entsprechenden Zellen verbunden.

Bei der Fallstudie eines hochintegrierten Schaltkreises, der mit dieser Entwurfsmethodik realisiert wurde, wird die zu erreichende Regularität des Entwurfs dadurch deutlich, daß zwischen den Register-, den Operationszellen und der Steuerlogik keine Verdrahtung auftritt, sondern die Zellen dieser Teile ineinander übergehen (siehe Abschnitt 2.3.4).

Ziele der Entwurfsmethodik

Das Hauptziel einer jeden Entwurfsmethode ist, den Entwickler so zu unterstützen, daß ausgehend von einer Spezifikation des Entwurfs Schaltkreise entwickelt werden, die bestimmten qualitativen Kriterien genügen.

Eine Entwurfsmethode kann deshalb daran gemessen werden, inwieweit sie diese Ziele unterstützt. Die dieser Arbeit zugrunde liegende Entwurfsmethode geht von synchronen Schaltkreisen aus, welche hierarchisch strukturiert und testbar sind. Dies erfordert ein vollständig integriertes Entwurfssystem, das auf einer für alle Beschreibungsebenen gemeinsamen Datenbasis operiert.

Modularität

Ähnlich den Methoden des Software-Engineerings ist auch für die Schaltungsentwicklung Modularität des Entwurfs der Schlüssel zu einer Wiederverwendbarkeit von Teilentwürfen. Sie ist eine wesentliche Voraussetzung zur Reduktion des Entwurfsaufwandes. Eine Zerlegung des Entwurfs in Teilentwürfe überschaubarer Komplexität ermöglicht die Kooperation mehrerer Entwickler und ist deshalb für die Entwicklung großer Systeme unabdingbar.

Änderungsfreundlichkeit

Über die Anforderungen bei Software-Erstellungssystemen hinaus ist unter Änderungsfreundlichkeit bei VLSI-Entwurfsmethoden nicht nur der Aufwand für Modifikationen des Entwurfs bei sich ändernder Spezifikation, sondern die Lokalität der Auswirkungen einer Änderung zu verstehen. 'Differentielle Entwurfs-Algorithmen' (LIPP 82) bewirken nur lokale Änderungen der Realisierung eines Entwurfs auf der Layout-Ebene, wenn in der funktionalen Beschreibung nur geringfügige Modifikationen vorgenommen wurden. Diese Proportionalität der Änderungen ist beim Schaltungsentwurf von besonderer Bedeutung, da bei Veränderung geometrischer Abmessungen in der Realisierung ein vollständiger Neuentwurf notwendig werden kann. Entwurfsmethoden, die z.B. die Verwendung programmierbarer Logik (PLA, ROM) besonders unterstützen, werden bei Änderungen auf der Register-Transfer Ebene geringere Modifikationen auf der Layout-Ebene nach sich ziehen als solche mit frei verdrahteter Logik (random logic).

Regularität

Die Regularität bezieht sich sowohl auf die gleichartige Verwendung

eines Teiles an verschiedenen Stellen eines Entwurfs und die räumliche Wiederholung von Elementen als auch auf einen sich zyklisch wiederholenden Ablauf des Algorithmus. Die Beschränkung der Entwurfsmethode auf synchrone Schaltwerke erzwingt eine zeitliche Regularität des Entwurfs, die geforderte Modularität und entsprechende Kontrollstrukturen der verwendeten Entwurfssprache fördern die strukturelle Regularität (vergleiche Abschnitt 4.3).

Testbarkeit

Die Unterstützung der Testbarkeit eines integrierten Schaltkreises durch die Entwurfsmethode ist für die Qualitätskontrolle bei der Herstellung des fertigen Bausteins von großer Bedeutung. Eine ausreichende Testüberdeckung kann nur erreicht werden, wenn bereits während des Entwurfs testfreundliche Strukturen gewählt wurden und die Entwurfswerkzeuge die Einhaltung von Grundregeln für die Konstruktion testfreundlicher Schaltungen erzwingen (vergleiche Abschnitt 1.2.7 und 7.2).

1.1.3. Zellkonzept und Zellbibliothek

Abhängig von den Anforderungen an die Leistungsfähigkeit sowie die Entwurfs- und Fertigungskosten eines hochintegrierten Schaltkreises werden verschiedene Entwurfstechniken angewandt. Der wesentliche Unterschied ist dabei die Komplexität der verwendeten elementaren Funktionsblöcke (Zellen). Bei Full-Custom-Technik (Kundenschaltkreise) wird jede Zelle für die spezielle Anwendung entwickelt und optimiert, während bei den Semi-Custom-Techniken allgemein verwendbare Zellen unabhängig von der aktuellen Anwendung entworfen und in einer Datenbasis (Zellbibliothek) gespeichert werden. Der Entwurf der zu fertigenden Schaltung erfolgt dann durch Verwendung der vordefinierten Zellen gleich einem Baukastensystem. Je nach der Komplexität und dem Standardisierungsgrad der Zellen unterscheidet man Gate Array, Standard-Zellen, allgemeine Zellen und Makro-Zellen-Technik.

Ein Halbleiterhersteller wird beim Entwurf eines leistungsfähigen Mikroprozessors, der in großen Stückzahlen gefertigt wird, beträchtlichen Aufwand für die Entwicklung eines Kundenschaltkreises akzeptieren, wenn dadurch ein kompaktes Layout mit hoher Schaltgeschwindigkeit erreicht

wird. In vielen anderen Fällen werden die geringeren Stückzahlen oder der frühe Einsatzzeitpunkt des Produkts eine langwierige und kostenintensive Entwicklung nicht ratsam erscheinen lassen. In diesen Fällen wird die Verwendung einer Semi-Custom-Technik naheliegen. Den hohen Entwicklungskosten eines Kundenschaltschaltkreises steht bei großen Stückzahlen eine erhebliche Einsparung an Fertigungskosten pro Baustein gegenüber, da ein Standardzellen-Entwurf etwa die doppelte Fläche benötigt, die der gleiche Entwurf als Kundenschaltschaltkreis haben würde. Der gleiche Entwurf als Gate-Array würde sogar den dreifachen Flächenbedarf aufweisen.

Um eine Einordnung des hier angestrebten Entwurfsverfahrens zu ermöglichen, werden im folgenden die verschiedenen Techniken und ihre Anwendungsgebiete diskutiert.

Kundenschaltkreise

Der Entwurf eines Kundenschaltschaltkreises ermöglicht bei Verwendung frei verdrahteter Logik die Realisierung sehr schneller oder äußerst kompakter Schaltkreise, abhängig vom Ziel der angewandten Optimierung. Der Entwicklungsaufwand ist, verglichen mit anderen Techniken, jedoch ungleich größer, da zur Erzielung einer hohen Packungsdichte der Transistoren und Verbindungen keine rechnergestützten Platzierungs- und Verdrahtungsalgorithmen eingesetzt werden können. Entwurfswerkzeuge können nur zur Simulation, zur Überprüfung der Entwurfsregeln (siehe 1.2.5) und zum Test eingesetzt werden. Die Kompaktheit des Layouts kann bei notwendigen lokalen Änderungen den teilweisen Neuentwurf des Schaltkreises notwendig machen, da zusätzliche Transistoren oder Gatter meist nicht ohne weiteres eingefügt werden können.

Die Ziele einer modernen Entwurfsmethodik werden beim Entwurf von Kundenschaltschaltkreisen insbesondere im Hinblick auf die Regularität des Entwurfs und seine Änderungsfreundlichkeit nicht immer erfüllt. Der Mikroprozessor Z8000 mit etwa 17.500 Transistoren ist die letzte bekannte vollständig als Kundenschaltschaltkreis entworfene Entwicklung. Neuere Mikroprozessoren enthalten bereits größere Anteile regulärer Strukturen, lediglich zeitkritische Komponenten werden noch in der Technik des Kundenschaltschaltkreises entworfen.

Gate Arrays

Die einfachste standardisierbare Struktur ist eine Zelle unverdrahteter Transistoren. Gate Arrays bestehen aus matrixförmig angeordneten Transistor-Zellen, aus denen durch eine zellinterne Verdrahtung einfache Gatter abgeleitet werden können (Bild 1.2). Werden diese Gatter durch eine externe Verdrahtung untereinander verbunden, entstehen Netze von Gattern (Bild 1.3.).

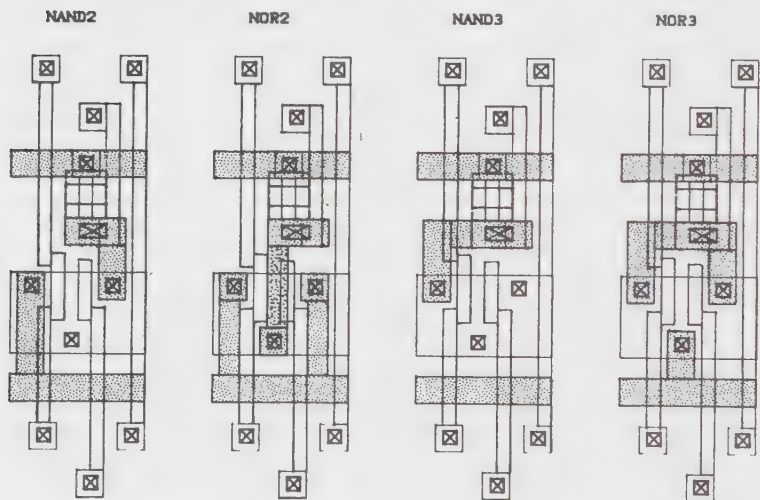


Bild 1.2.: Transistorzellen mit und ohne interne Verdrahtung

Während bei Kundenschaltschaltungen der gesamte Schaltkreis nach dem für die spezielle Anwendung entworfenen Layout gefertigt wird, werden Gate Arrays bis auf die Verdrahtung vorgefertigt (master-slice, uncommitted logic array). Der Hersteller fertigt die Matrix der Transistoren sowie Ein/Ausgänge in sehr großen Stückzahlen und bringt erst in den letzten kundenspezifischen Verdrahtungsebenen die Metallisierungsschichten auf.

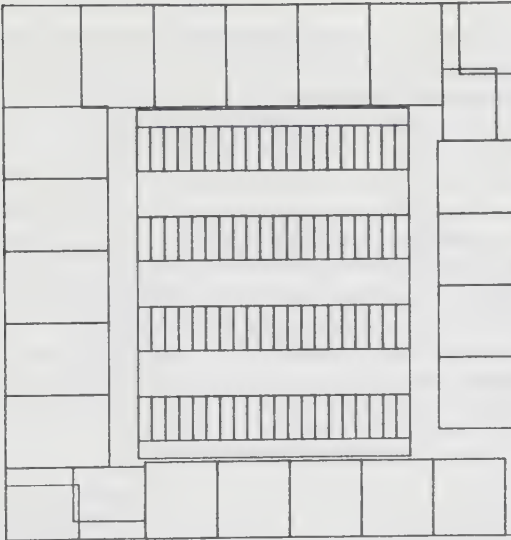


Bild 1.3.: Matrixanordnung bei Gate Arrays

Dies führt zu preisgünstigen Schaltkreisen, die jedoch nur eine beschränkte Komplexität aufweisen. Dies ist auf die zur Verdrahtung notwendigen Kanäle zwischen den Zellen und die nur teilweise mögliche Nutzung vorgefertigter Transistoren als Gatter (60-80%) zurückzuführen.

Der Einsatz von rechnergestützten Entwurfswerkzeugen ist bei Gate Arrays weit fortgeschritten, da lediglich Platzierungs- und Verdrahtungsprogramme benötigt werden (MOULLION 83, BURSKEY 82). Das Entflechtungsproblem reduziert sich auf die Realisierung der gewünschten Gatterfunktionen an geeigneter Stelle der Matrix und die Verbindung gemäß dem vorgegebenen Schaltplan. Um eine Beschränkung der Rechenzeiten bei Platzierung und Verdrahtung zu erreichen, werden nicht alle Zellen der Matrix ausgenutzt, da insbesondere bei nur einer Verdrahtungsebene eine planare Lösung nicht immer möglich ist.

Gate Arrays werden eingesetzt, wenn entweder geringe Stückzahlen zu erwarten sind oder die zu realisierende Schaltung eine geringe Komplexi-

tät besitzt. Der relativ einfache Entwurfsprozeß und die zur Aufbringung der Verdrahtung notwendigen wenigen Prozeßschritte ermöglichen darüber hinaus kurze Realisierungszeiten.

Standard-Zellen

Im Gegensatz zu Gate Arrays sind bei Standard-Zellen die elementaren Funktionsblöcke bereits vollständig entworfen und getestet. Dem Entwickler steht eine Sammlung standardisierter Zellen in einer Zellbibliothek zur Verfügung, welche die am häufigsten benötigten Zellen einschließlich ihrer Spezifikation enthält (LOESCH 82). Der Entwurf der Zellen, die Simulation, der Test und die Erstellung der Zellbibliothek wird vom Hersteller übernommen und reduziert dadurch die Kosten einer Schaltungsrealisierung erheblich.

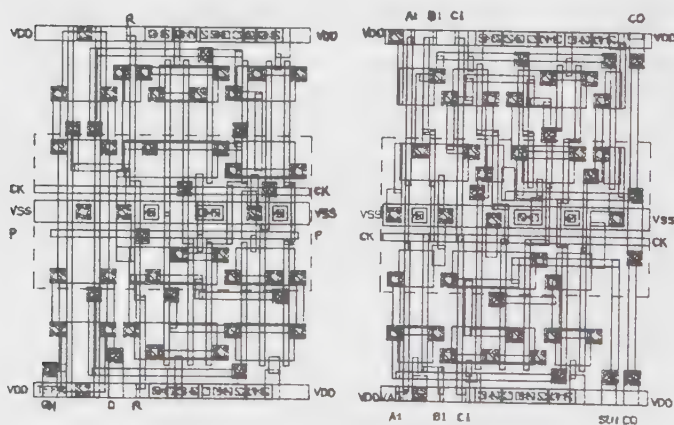


Bild 1.4.: Standardzelle für zweiseitige Verdrahtung

Beim Entwurf eines Schaltkreises in Standard-Zellen-Technik haben die Zellen bei unterschiedlicher Breite eine einheitliche Höhe und Grundstruktur. Speziell die Leitungen zur Stromversorgung sind so angeordnet,

daß bereits durch die Aneinanderreihung der Zellen eine Verbindung entsteht. Jede Zelle hat üblicherweise an der Ober- und Unterkante identische Anschlüsse, um eine einfache Verdrahtung zu ermöglichen (Bild 1.4.). Die Zellen werden zeilenförmig aneinandergereiht und gegenseitig verbunden. Dazu wird zwischen den Zeilen ein Verdrahtungskanal freigehalten, dessen Breite sich nach der Anzahl der parallel zu legenden Leitungen richtet.

Bei älteren Standardzell-Systemen sind die Anschlüsse nur an einer Seite der Zellen herausgeführt, so daß jeweils zwei Zellreihen zusammengefügt werden und eine gemeinsame Spannungsleitung haben (BIRZELE 82). Dies erschwert jedoch die globale Verdrahtung, da ein nach unten weisender Anschluß unter Umständen um die Zelle herum nach oben geführt werden muß. Die Verdrahtung der Zellen erfolgt im Verdrahtungskanal zwischen den Zellen, wobei zur Auflösung von Kreuzungen teilweise ein Wechsel der Verdrahtungsebenen erfolgen muß. Verbindungen von einem Kanal zum nächsten werden entweder zwischen den Zellen (Bild 1.5.) oder durch Zellen hindurch vorgenommen. Dazu können bei zweiseitig verdrahtbaren Zellen nicht belegte Anschlüsse verwendet werden, die von der einen Seite der Zelle auf die andere führen.

Eine Variation der Standard-Zellen-Technik sieht Zellen unterschiedlicher Höhe vor, wobei jedoch in jeder Zeile Zellen gleicher Höhe aneinandergereiht werden (augmented standard-cell SCHWEIKERT 83).

Die Anwendung der Standard-Zellen-Technik erfordert gegenüber Kundenschaltschaltkreisen ein mehrfaches der Chip-Fläche. Dies liegt einerseits an dem größeren Platzbedarf der Zellen durch zweiseitige Anschlüsse, zum anderen an den Verdrahtungskanälen, deren Höhe durch den maximalen Platzbedarf der Verdrahtung an einer Stelle des Kanals bestimmt wird. Die Verwendung bereits fertiggestellter Zellen und der Einsatz von Platzierungs- und Verdrahtungsprogrammen ermöglichen jedoch einen wesentlich einfacheren und schnelleren Entwurf.

Im Gegensatz zu Gate Arrays müssen zur Herstellung des Schaltkreises alle Masken nach dem kundenspezifischen Layout hergestellt und der vollständige Fertigungsprozeß durchlaufen werden. Dies ist bei größeren Stückzahlen jedoch wirtschaftlich, da Standard-Zellen wesentlich komplexere Entwürfe erlauben, beziehungsweise geringere Flächen benötigen.

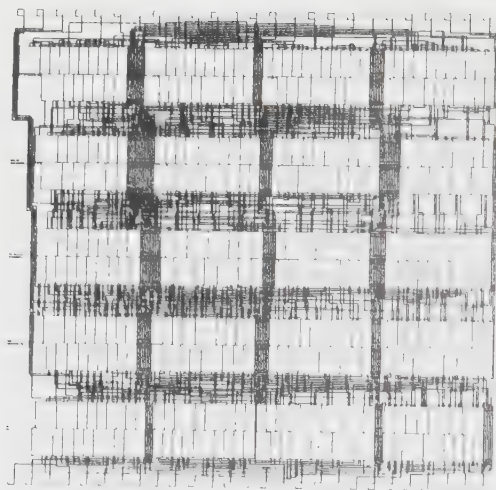


Bild 1.5.: Verdrahtung einseitig verdrahtbarer Zellen

Allgemeine Zellen

Zellen unterschiedlicher Höhe und Breite erlauben eine bessere Ausnutzung der Chip-Fläche als die Standard-Zellen-Technik, bei der die Beschränkung auf Zellen gleicher Höhe zu einem nicht optimalen Layout einzelner Zellen führt. Die strenge zeilenförmige Anordnung mit relativ breiten Verdrahtungskanälen erfordert im allgemeinen einen größeren Flächenbedarf, als die bei allgemeinen Zellen mögliche freie Platzierung mit direkter Verdrahtung (Bild 1.6). Darüber hinaus ist es bei allgemeinen Zellen sinnvoll mehrere Zellen zu einem komplexen Funktionsblock zusammenzufassen, der seinerseits wiederum als allgemeine Zelle aufgefaßt werden kann. Damit ist es möglich, einen modularen Entwurf zu erstellen.

Entscheidenden Einfluß auf die Kompaktheit des Entwurfs hat die Güte der Platzierung (WIPFLER 83), die so optimiert wird, daß die Gesamtlänge der

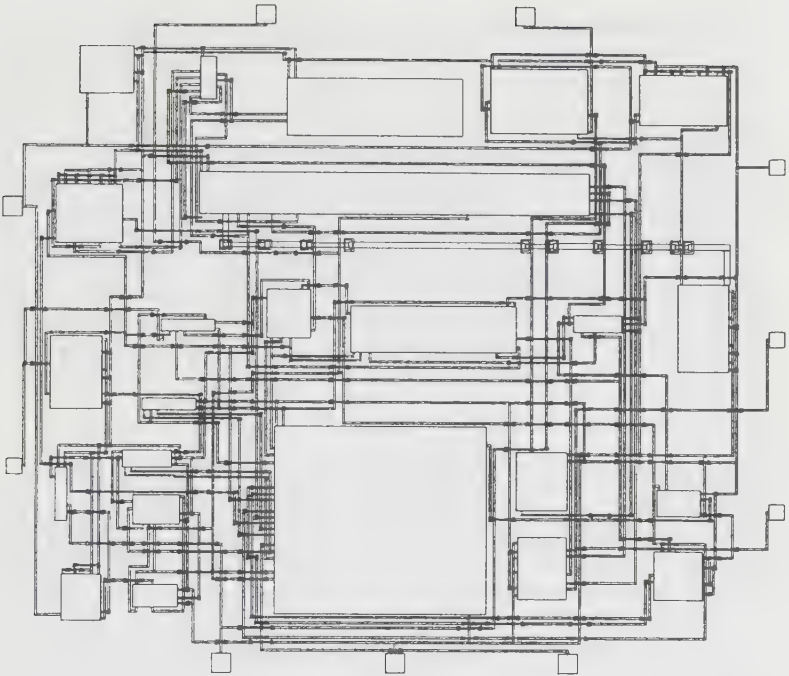


Bild 1.6.: Platzierung und Verdrahtung allgemeiner Zellen

Verdrahtung minimiert wird. Der Verdrahtungsalgorithmus verwendet die bei der Platzierung entstandenen Kanäle und löst Kreuzungen verschiedener Leiterbahnen durch einen Wechsel der Verdrahtungsebenen. Als schwierig erweist sich dabei die Verdrahtung der Stromversorgungsnetze, die eine planare Einbettung erfordern, da in vielen Fällen nur eine metallische Verdrahtungsebene zur Verfügung steht (ROTHERMEL 83).

Makro Zellen

Eine Erweiterung der allgemeinen Zellen auf Zellen variabler Dimension und parametrisierbarer elektrischer Eigenschaften sind die Makro Zellen. Ziel ist eine automatische Generierung von Zellen aus einem vorhandenen Layout mit vorgegebener maximaler Dimension und vom Entwickler vorgegebenen elektrischen Eigenschaften. Ausgehend von einer Strukturbeschreibung der Makrozelle wird durch verschiedene Algorithmen die aktuelle Dimension der Zelle ermittelt. Als Parameter für die Generierung werden dabei Entwurfsregeln für die Transistoren, maximal zulässige Verzögerungszeit, Position der Zellanschlüsse und elektrische Eigenschaften vorgegeben (VERGNIERES 80). Die Platzierung und Verdrahtung der generierten Zellen erfolgt dann nach den gleichen Prinzipien wie bei allgemeinen Zellen.

Zellbibliothek

Voraussetzung für die Anwendung einer Semi-Custom-Technik ist die Existenz einer Zellbibliothek, in der die als Bausteine verwendeten Zellen gespeichert sind. Diese wird im allgemeinen vom Hersteller in mehreren Technologien mit äquivalenten Zellentwürfen zur Verfügung gestellt um dem Anwender hinsichtlich Stromverbrauch, Flächenbedarf und Leistung verschiedene Alternativen zu bieten. Die Komplexität der Zellen reicht dabei vom einfachen Transistor über logische Gatter, dynamische und statische Register bis zu Zählern, Schieberegistern, Vergleichen, Addierer und Multiplexern. Sogar umfassende Schaltkreise wie Speicher und einfache Mikroprozessoren sind verfügbar.

1.2. STRUKTUR RECHNERGESTÜTZTER ENTWURFSSYSTEME

Die Integration von Werkzeugen der Bereiche System- und Schaltungsentwurf zu einem einheitlichen Entwurfssystem ist wesentliche Voraussetzung für die Entwicklung moderner Rechnergenerationen (NEWTON 81). Rechnergestützte Entwurfshilfsmittel ermöglichen eine effiziente und fehlerfreie Durchführung von Aufgaben mit einem hohen Anteil sich wiederholender Tätigkeiten. Ein bekanntes Beispiel ist die logische und elektrische Simulation von Schaltkreisen, die ohne Rechnerunterstützung kaum durchzuführen ist. Die Komplexität von VLSI-Schaltkreisen erfordert die Implementierung effizienter Algorithmen, da selbst bei der Beschränkung der Simulation auf Teile des Entwurfs erhebliche Rechenzeiten notwendig sind.

Die Entwicklung von rechnergestützten Entwurfswerkzeugen ist langwierig und kostenintensiv. Deshalb ist es notwendig, Programme zu entwickeln, die breitere Anwendung finden können und nicht vor ihrer Fertigstellung durch technologische Fortschritte überholt werden.

1.2.1. Systeme und Programme

Die heute verfügbaren Programme unterstützen häufig bestimmte Phasen des Entwurfsprozesses und sind auf die weiter verbreitete Gate-Array oder Standard-Zellen-Technik beschränkt (eine Übersicht kommerzieller Systeme enthält CLEEMPUT 83).

Schwerpunkt dieser Arbeit ist, die theoretischen Grundlagen für die Gestaltung und Implementierung einer gemeinsamen Datenbasis für alle Entwurfswerkzeuge und die Konzeption eines integrierten Synthesystems von der Architektur- bis zur Layoutebene zu erarbeiten. Da bis heute kein geschlossenes Entwurfssystem für Kundens Schaltkreise verfügbar ist, das alle Phasen des Entwurfs unterstützt, wurde in Erlangen eine Entwurfsumgebung aufgebaut, die aus verschiedenen Programmsystemen besteht, die jeweils eine Phase des Entwurfsprozesses unterstützen. Da die einzelnen Programme teilweise verschiedenen Quellen entstammen,

bilden sie kein geschlossenes System mit abgestimmten Schnittstellen und gemeinsamer Datenbasis. Sie sind vielmehr auf die Erfordernisse der einzelnen Beschreibungsebenen zugeschnitten und erfordern bei fortschreitender Detaillierung des Entwurfs mit dem damit verbundenen Übergang zur nächst niederen Beschreibungsebene eine Neuformulierung. In Bild 1.7 ist eine Übersicht und Zuordnung der einzelnen Programme zu den entsprechenden Beschreibungsebenen dargestellt.

ENTWURFS-WERKZEUGE

	GPSS-F	ERES	DISIM	SPICE/MSINC	LUCIE	SUPREM
SYSTEM						
REGISTER-TRANSFER						
GATTER						
TRANSISTOR						
LAYOUT						
TECHNOLOGIE						

Bild 1.7.: Erlanger Entwurfswerkzeuge

Der Entwurf eines hochintegrierten Schaltkreises kann nicht losgelöst von der Systemumgebung, in der er eingesetzt werden soll, vorgenommen werden. Das Programmpaket GPSS-F (SCHMIDT 78) ermöglicht die Simulation des gesamten Systemverhaltens auf der Grundlage von Warteschlangennetzen und dient einer Bewertung und Optimierung der zu erwartenden Leistung des Gesamtsystems.

Den Kern des Entwurfssystems bilden jedoch die Programme der Register-Transfer-, Gatter-, Transistor- und Layoutebene. Die Rechner-Entwurfsprache ERES (GARDILL 77, BRENDDEL 82) ist eine Register-Transfersprache

mit exakter Zeitbeschreibung und ermöglicht neben der Spezifikation des Entwurfs auch dessen funktionale Simulation (siehe Abschnitt 1.2.3.). Programme zur Fehlersimulation auf der Gatterebene (DISIM siehe 1.2.7.) und zur elektrischen Simulation auf der Transistorebene (SPICE, MSINC siehe 1.2.6.) ergänzen die funktionale Simulation.

Ein typisches Beispiel eines Layout-Entwurfssystems für Kundenschaltkreise ist das bei ENSIMAG Grenoble entwickelte System LUCIE ('Language Universitaire pour la Conception de Circuits Integres pour l'Enseignement' GUYOT 81), das besonders an französischen und italienischen Universitäten größere Verbreitung gefunden hat. Die Programme wurden im Rahmen mehrerer Studienarbeiten ergänzt und an vorhandene Grafikperipherie angepaßt. Die beim Entwurf des Layouts verwendeten Programme umfassen einen interaktiven, graphischen Editor (SCHRÖDER 84) und ein Plotprogramm zur Erzeugung einer Zeichnung des entworfenen Layouts (FILLER 84). Eine Überprüfung der Entwurfsregeln ermöglicht die automatische Erkennung von Verletzungen geometrischer Entwurfsregeln (ZINK 84, siehe 1.2.5.). Die Netzwerkbeschreibung als Eingabe für eine elektrische Simulation mit MSINC kann mit Hilfe eines Schaltkreis-Extraktors gewonnen werden.

Die allen Programmen gemeinsame Datenbasis besteht aus Zellbeschreibungen, die im wesentlichen geometrische Daten des Layouts enthalten. Diese werden in Form von Rechtecken verschiedener technologischer Maskenebenen durch die Layout-Sprache LUCIE definiert. Speziell die Daten für die elektrische Simulation und die Prüfprogramme müssen aus den geometrischen Daten berechnet werden und sind nicht in der Datenbasis enthalten. Für ein ebenenübergreifendes Entwurfssystem ist deshalb die Konzeption einer gemeinsamen Datenbasis, die alle Entwurfsdaten der verschiedenen Beschreibungsebenen beinhaltet, von besonderer Bedeutung.

In den folgenden Abschnitten werden die wesentlichen Eigenschaften der in den verschiedenen Phasen des Entwurfs eingesetzten Werkzeuge und Methoden im Hinblick auf ihre Verwendung in einem integrierten Entwurfssystem mit Syntheseeigenschaften beschrieben.

1.2.2. Entwurfssprachen

Die Architektur und Leistungsfähigkeit eines Entwurfs werden in der

Systemspezifikation beschrieben. Die Implementierung der Architektur durch Komponenten wie Register, Speicher und Busse sowie die Aufteilung in verschiedene Teilsysteme liefert eine strukturelle Beschreibung des Entwurfs. Eine formale Beschreibung eines Entwurfs muß neben der Funktion und der Struktur auch zeitliche Randbedingungen umfassen, da diese eine Bewertung der Leistungsfähigkeit erst ermöglichen. Die Form der Beschreibung ist dabei untrennbar mit der zugrunde liegenden Entwurfsmethode verknüpft. Die formale Beschreibung eines zu realisierenden Entwurfs ist der erste Schritt in einer Reihe von aufeinanderfolgenden Transformationen, die letztlich zu einem Schaltkreis führen sollen, der den Zielen der Entwurfsmethode genügt. Dabei ist eine exakte Definition der Semantik der in den einzelnen Phasen verwendeten Beschreibungsmittel unumgänglich, wenn ein integriertes Entwurfssystem angestrebt wird.

Eine Automatisierung des Entwurfsprozesses ist nur dann erfolgversprechend, wenn bereits die funktionale Spezifikation rechnergestützt durchgeführt werden kann. Die formale Beschreibung dient nicht nur Dokumentationszwecken, sondern ist Grundlage für eine bereits in der Konzeptionsphase notwendige Bewertung des Entwurfs. Neben der Simulation von funktionalem und zeitlichem Verhalten können aufgrund einer formalen Beschreibung des Entwurfs bereits Aussagen über den zu erwartenden Platzbedarf bei einer Realisierung als integrierter Schaltkreis gemacht werden.

Die Beschreibung eines Entwurfs auf der Register-Transferebene durch Rechner-Entwurfssprachen hat eine lange Tradition (eine Literaturübersicht enthält CLEEMPUT 76). Die programmiersprachliche Darstellung eines Entwurfs ermöglicht sowohl die funktionale als auch die strukturelle Beschreibung. Dabei werden bei fortschreitender Detaillierung des Entwurfs funktionale Beschreibungen zunehmend durch strukturelle ersetzt, indem Funktionen wie arithmetische oder komplexe logische Operationen durch die strukturelle Beschreibung ihrer Realisierung mit Hilfe primitiver Elemente ersetzt werden. Die Spezifikation des zeitlichen Verhaltens bleibt mit zunehmender Detaillierung stets erhalten.

Graphische Methoden, wie sie auf dem Gebiet der Schaltungsentwicklung vorherrschen, beinhalten nur strukturelle Eigenschaften und sind für eine funktionale Beschreibung der Architektur weniger geeignet. So enthält der Schaltplan eines Entwurfs bereits die Realisierung einer

Funktion, während eine abstrakte Spezifikation als Automat, wie sie mit einer Rechner-Entwurfssprache möglich ist, die Realisierung nicht vorwegnimmt.

Die Abstraktion der in einer programmiersprachlichen Formulierung verwendeten Konstrukte gegenüber der Realisierung durch die Hardware hat bisher eine breitere Anwendung der Rechner-Entwurfssprachen verhindert. Eine enge Bindung der Elemente einer formalen Beschreibung an die ihnen zugeordneten Modelle der Realisierung ermöglicht jedoch eine exakte Definition der zu realisierenden Schaltung. Die Semantik der auf der Register-Transferebene verwendeten Rechnerentwurfssprache muß deshalb auf den elementaren technologischen Komponenten basieren. Die in späteren Kapiteln definierte Rechner-Entwurfssprache CERES ist eine, auf den Entwurf integrierter Schaltkreise zugeschnittene, Erweiterung von ERES (GARDILL 77) und enthält Konstrukte, die einerseits typische Elemente wie statische und dynamische Register, Transferegatter, Speicher und logische Verknüpfungen modellieren, andererseits jedoch auch die Modularisierung des Entwurfs und die hierarchische und mehrfache Verwendung von Teilsystemen gestatten, ohne dadurch von einer bestimmten Technologie abhängig zu werden.

1.2.3. Funktionale Simulation

Die funktionale Simulation erlaubt bereits während der Spezifikationsphase eine Überprüfung der Korrektheit der Datenpfade und Operationen sowie des zeitlichen Verhaltens eines Entwurfs. Dabei werden neben Fehlfunktionen auch Zeitkonflikte wie mehrfaches Laden eines Registers aus verschiedenen Quellen oder Laufzeiteffekte in logischen Verknüpfungen aufgedeckt, wenn die Entwurfssprache die Deklaration von zeitlichen Attributen erlaubt.

Simulation kann auf allen Beschreibungsebenen durchgeführt werden. Wesentlich für den Entwurf hochintegrierter Schaltkreise sind jedoch die funktionale Simulation auf der Register-Transferebene und die elektrische Simulation auf der Transistorebene (Abschnitt 1.2.6.). Der funktionalen Simulation während der Spezifikation eines Entwurfs kommt dabei besondere Bedeutung zu, da Entwurfsfehler, welche nicht bereits in der ersten Phase des Entwurfsprozesses entdeckt werden, einen zumindest teilweisen Neuentwurf notwendig machen können. Da bei integrierten

Schaltkreisen eine Fertigung von Testmustern der Schaltung im Gegensatz zu konventionellen Techniken aus Kostengründen unterbleibt, wird die Simulation zum wichtigsten Werkzeug der Entwicklung. Sie hat die Aufgabe, vom Entwickler vorgesehene Verhaltensweisen zu überprüfen und eventuell vorhandene Entwurfsfehler aufzudecken. Eine möglichst umfassende Simulation des Entwurfs ist deshalb unumgänglich, was eine effiziente Implementierung des Simulators voraussetzt. Rechner-Entwurfssprachen mit entsprechenden Simulatoren erlauben heute die Beschreibung so komplexer Schaltwerke wie mikroprogrammierter Prozessoren einschließlich der zugehörigen Mikroprogramme.

Die Spezifikation mit Hilfe einer Rechner-Entwurfssprache ist eine Approximation des tatsächlichen Verhaltens. Simulationsergebnisse sind deshalb nur dann hinreichend zuverlässig, wenn das zugrunde liegende Modell die wesentlichen Prinzipien sequentieller, digitaler Schaltwerke korrekt nachbildet und zudem zeitliche Toleranzen der Technologie berücksichtigt. Die Genauigkeit wird dabei nicht prinzipiell durch die Anzahl der logischen Zustände verbessert, da Schwankungen in der Güte des Fertigungsprozesses sich insbesondere auf das zeitliche Verhalten einer integrierten Schaltung auswirken. Die Zuverlässigkeit einer funktionalen Simulation erfordert vielmehr eine globalere Abschätzung des zeitlichen Verhaltens als dies bei der Verwendung der Ergebnisse der detaillierten elektrischen Simulation möglich wäre.

Die gleichzeitige Simulation eines Entwurfs auf verschiedenen Beschreibungsebenen mit unterschiedlicher Genauigkeit wird als mixed-mode Simulation bezeichnet (DEMAN 82). Dabei werden kritische Teile des Entwurfs auf der Transistorebene, die Umgebung auf der Gatter- oder Register-Transferebene simuliert. Für Systemteile, die eine genauere Simulation erfordern, wird eine Transientenanalyse durchgeführt, während die restliche Schaltung logisch simuliert wird. Der Rechenaufwand für die Simulation der gesamten Schaltung wird dadurch wesentlich geringer als bei einer vollständig elektrischen Simulation. Die Simulation kritischer Teile des Entwurfs beruht auf technologieabhängigen Makromodellen und erfordert die Angabe prozeßabhängiger Parameter sowie die Spezifikation der Flächen von Polysilizium- und Diffusionsgebieten zur Berechnung parasitärer Kapazitäten. Durch diese Technologieabhängigkeit ist die mixed-mode Simulation eher eine Verbesserung der konventionellen elektrischen, als ein Ersatz für die funktionale Simulation.

1.2.4. Layout

Der Entwurf des Layouts für integrierte Schaltkreise ist im Entwurfsprozeß die zeitaufwendigste und am stärksten mit Fehlern behaftete Phase. Sehr frühzeitig wurden deshalb rechnergestützte Systeme für die Layoutentwicklung und Überprüfung auf Einhaltung von Entwurfsregeln entwickelt (SOUKUP 81). Die Entwicklung preiswerter graphischer Arbeitsplätze mit hochauflösender Farbrastergraphik hat zu einer weiten Verbreitung von interaktiven Systemen für den Entwurf hochintegrierter Schaltungen geführt.

Layout-Entwurfsmethodik

Die beim Entwurf der Fallstudie angewandten Prinzipien werden hier kurz dargestellt, da bei der Erstellung des Layouts eines integrierten Schaltkreises durch einen systematischen Entwurf der Topologie der einzelnen Zellen eine gute Ausnutzung der Chip-Fläche erreicht werden kann.

Für jede primitive Funktion der Schaltung wird eine Zelle entworfen, bei der Daten- und Steuerleitungen möglichst orthogonal angeordnet sind. Die Anschlußpunkte der Zellen und die Führung von Daten- und Steuerleitungen werden zuerst festgelegt, die Transistoren anschließend unter oder zwischen Datenleitungen angeordnet. Werden die orthogonalen Leitungen in zwei verschiedenen Ebenen (Aluminium und Polysilizium) realisiert, treten nur an wenigen Stellen Planaritätsprobleme auf. Benachbarte Zellen werden in ihren Anschlußpunkten so angepaßt, daß eine Verdrahtung durch direktes Zusammenfügen benachbarter Zellen erfolgen kann (wiring by abutment). Dadurch wird eine matrixförmige Aneinanderreihung der Zellen ohne zusätzliche Verdrahtungen ermöglicht.

Die Orthogonalität von Daten- und Steuerleitungen erlaubt die Anordnung der Steuerlogik ohne zusätzliche Verdrahtung seitlich vom Datenpfad, so daß der Gesamtflächenbedarf minimiert wird. Zugunsten dieser globalen Optimierung des Zell-Layouts wird auf eine lokale verzichtet, da der Platzbedarf für eine globale Verdrahtung im allgemeinen größer ist als der Gewinn bei der Optimierung einzelner Zellen.

Layout-Sprachen

Die Beschreibung eines Layouts in einer symbolischen Layoutsprache erfordert keine weiteren Hilfsmittel. Transistoren werden dabei durch Rechtecke oder Polygone beschrieben, die jeweils mit ihren bezüglich der umgebenden Zelle relativen Koordinaten und der zugehörigen Maskenebene definiert werden. Eine Zusammenfassung verschiedener Transistoren und Verbindungen zu einer Zelle erlaubt eine mehrfache Wiederverwendung bei regulären Strukturen. Die Inkarnationen dieser Zellen werden mit entsprechenden Parametern zur Translation, Rotation und Spiegelung aufgerufen und können dann zur Beschreibung komplexerer Zellen verwendet werden. Bekanntestes Beispiel einer Layout-Sprache ist CIF ('Caltech Intermediate Form' MEAD 80).

Die Hauptanwendung von Layout-Sprachen ist jedoch weniger die Layout-Beschreibung, als die Speicherung und der Transfer der geometrischen Daten in lesbarer Form. Beim Layout-Entwurf können somit bereits erstellte Zellen auf einfache Weise eingebunden werden. Die Erzeugung der Maskenbänder wird durch eine Transformation der Koordinaten und Dimensionen in die Metrik der Maskenfertigung und eine Ersetzung komplexer geometrischer Objekte durch Rechtecke vorgenommen.

Neuere Entwicklungen von Layoutsprachen verwenden eine nichtmetrische Definition des Layouts (SASIRY 82)

Neuere Entwicklungen von Layoutsprachen verwenden eine nichtmetrische Definition des Layouts (SASIRY 82), die sich besonders für das symbolische Layout eignet. Die Metrik eines Entwurfs wird dabei erst bei der Kompaktierung erzeugt, wodurch eine von den aktuellen Entwurfsregeln weitgehend unabhängige Beschreibung des Layouts erreicht wird.

Geometrisches Layout

Wurde anfänglich der Entwurf noch am Zeichentisch entwickelt und anschließend digitalisiert, geht man heute zum interaktiven, graphischen Entwurf am Bildschirm über, indem die verschiedenen Maskenebenen oder Symbole für die primitiven Elemente und ihre Verbindungen eingegeben werden. In beiden Fällen werden rein geometrische Daten gespeichert, aus

denen sich das am Bildschirm erscheinende Layout ableitet. Diese können dann zur Überprüfung geometrischer und elektrischer Entwurfsregeln ausgewertet werden.

Eine Variante des geometrischen Layouts sind Strichdiagramme, die jedoch nicht für die interaktive Erstellung, sondern zur Dokumentation der Layoutstruktur geeignet sind, da für die Transistoren keine Dimensionierung angegeben und nur die topologische Struktur des Layouts gezeichnet wird (MEAD 80).

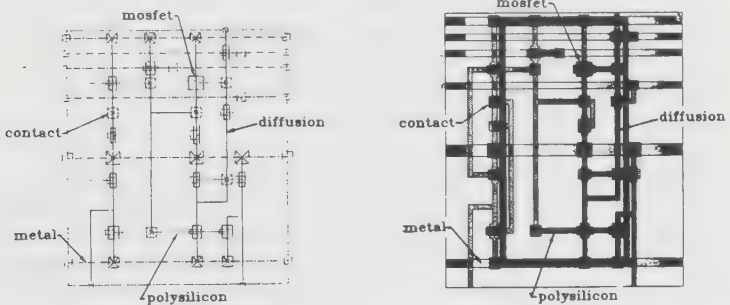


Bild 1.8.: Symbolisches und erweitertes Layout

Symbolisches Layout

Bei Systemen zur Erstellung eines symbolischen Layouts (NEWTON 81) werden primitive Schaltungselemente, wie minimale Last-, Transfer- und Treibertransistoren, als Symbole eingegeben und die Transistordimensionen, wenn notwendig, an die Erfordernisse der Schaltung angepaßt. Die Verbindungen werden durch einfache Striche symbolisiert, die den Maskenebenen durch verschiedene Darstellungsarten zugeordnet werden (Bild 1.8). Komplexe Layout-Regeln wie Leitungsstärke und minimaler Abstand werden dabei nicht berücksichtigt. Die gegenüber einem vollständigen Layout übersichtlichere Darstellung ist einfacher zu ändern, da lediglich die Kreuzungsfreiheit von Leitungen der gleichen Maskenebene zu berücksichtigen ist.

Das erstellte symbolische Layout wird dann automatisch kompaktiert, indem die Verbindungsleitungen unter Einhaltung der technologischen Entwurfsregeln so modifiziert werden, daß der Flächenbedarf einer Zelle minimiert wird (KELLER 82). Dabei werden iterativ alle Leitungen gekürzt und lokale Modifikationen wie Rotation einzelner Symbole und Abknickung von Verbindungen durchgeführt. Eine automatische Kompaktierung des symbolischen Layouts erreicht nicht immer die Dichte eines von Hand entworfenen geometrischen Layouts, reduziert jedoch den Aufwand für die Erstellung wesentlich.

1.2.5. Entwurfsregeln und topologische Analyse

Das Layout eines Schaltkreises muß gewissen geometrischen Entwurfsregeln (design rules) genügen, um die Funktionsfähigkeit zu garantieren. Dabei wird unterschieden zwischen geometrischen Entwurfsregeln, welche Fertigungstoleranzen berücksichtigen, und elektrischen, die Bedingungen für die Treiberleistung der Transistoren festlegen. Diese Entwurfsregeln sind grundsätzlich sowohl vom Herstellungsprozeß (NMOS, CMOS, I²L, bipolar...), als auch von der speziellen Prozeßlinie des Herstellers abhängig. Herstellerunabhängige Entwurfsregeln können deshalb nicht von minimalen Randbedingungen eines bestimmten Prozesses ausgehen, sondern sind großzügiger auszulegen, wodurch nicht die maximale Integrationsdichte ausgenutzt werden kann, die ein bestimmter Prozeß einer speziellen Prozeßlinie erreichen kann.

Geometrische Entwurfsregeln

Die geometrischen Entwurfsregeln legen die minimal zulässigen Größen für Bauelemente und Abstände zwischen den Leiterbahnen fest. Sie werden durch physikalische Effekte zwischen Gebieten unterschiedlicher Dotierung und Materialien sowie die Grenzen der verwendeten Photolithographie bestimmt. Zusätzlich sollen die bei der Maskenfertigung auftretenden Verzeichnungen und die während der Schaltkreis-Herstellung unvermeidlichen Verformungen der Siliziumscheibe kompensiert werden.

Um allgemeine geometrische Entwurfsregeln definieren zu können, die in weiten Bereichen der Skalierung gelten, wurde von C.Mead und L.Conway

das technologie- und prozeßunabhängige Längenmaß LAMBDA eingeführt (MEAD 80). Das in LAMBDA-Einheiten definierte Layout eines Entwurfs wird vor der Maskenfertigung in eine herstellerabhängige Metrik transformiert.

Entwurfsregeln der Fallstudie

Die für die Fallstudie des zweiten Kapitels verwendeten geometrischen Entwurfsregeln (GUYOT 83) sind eine erweiterte und verbesserte Version der in (MEAD 80) angegebenen. Die graphische Darstellung der einzelnen Ebenen des NMOS-Prozesses wurde jedoch gleich gewählt (Diffusion grün, Polysilizium rot, Aluminium blau, Implantation gelb, Kontakte schwarz). Insbesondere wurden vergrabene Kontakte (buried contact, eine Diagonale) aufgenommen, da der verwendete Prozeß diese zuließ. Im Gegensatz zu den mit einer Aluminiumschicht bedeckten Kontakten (contact cut, zwei Diagonalen) können vergrabene Kontakte zwischen Diffusion und Polysilizium ohne Beeinträchtigung unter Aluminium-Leiterbahnen platziert werden, was ein kompakteres Layout ermöglicht. Die Dimensionierung der Abstände, Breiten und Überlappungen wurde in standardisierten LAMBDA Einheiten angegeben (MEAD 80), um von der tatsächlichen Integrationsdichte des verwendeten Prozesses unabhängig zu sein. Eine Transformation in die prozeßspezifische Dimensionierung ($\lambda = 2.25\mu\text{m}$) wurde erst vor der Maskenfertigung durchgeführt.

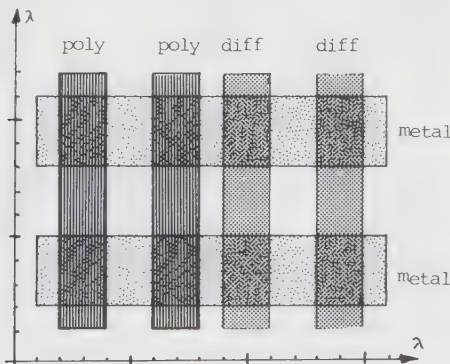


Bild 1.9.: Maskenspezifische Entwurfsregeln

Die Leistungsfähigkeit der Photolithographie, der verwendete Ätzprozeß und die Dotierungs- und Aufdampftechniken bestimmen minimale Größe und Mindestabstand von Bauelementen und Leiterbahnen. Die geometrischen Entwurfsregeln müssen gewährleisten, daß eventuell verbleibende Aluminiumreste nicht zu Kurzschlüssen zwischen Leiterbahnen, Einbuchtungen in Aluminiumleitern nicht zu Unterbrechungen führen, die teilweise erst während des Betriebs der Schaltung durch Elektromigration hervorgerufen werden könnten. Der Flächenwiderstand und die maximal zulässige Stromdichte bestimmen die Dimensionierung der Diffusions- und Polysiliziumgebiete. Diese Randbedingungen ergeben die in Bild 1.9 dargestellten, maskenspezifischen Entwurfsregeln.

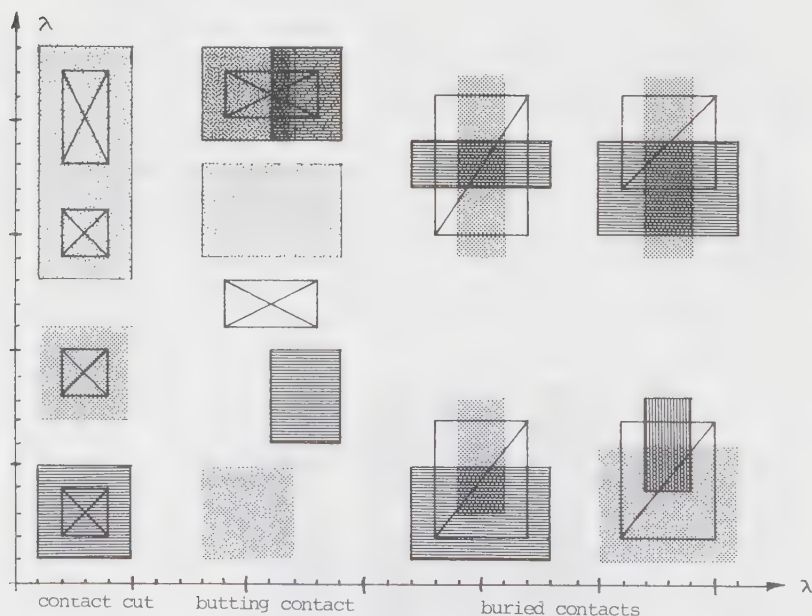


Bild 1.10.: Entwurfsregeln für Kontakte

Geometrische Randbedingungen zwischen mehreren Maskenebenen werden durch die Justiergenauigkeit der Masken und die, durch die Erhitzung bei der Diffusion hervorgerufenen Verformungen der Siliziumscheibe bestimmt. Die Größe der Kontaktlöcher wird durch den Kontaktlochwiderstand und die Ätzbarkeit kleiner Strukturen beeinflusst. Da hierbei mehrere Prozeßschritte aufeinander abgestimmt sein müssen, sind Überlappungen der einzelnen Ebenen notwendig (Bild 1.10, eine ausführliche Erläuterung der Kontakte und ihrer Funktion findet man in MEAD 80). Die aktiven Bauelemente wie Treiber- und Lasttransistoren sind von den Toleranzen des Herstellungsprozesses am stärksten betroffen. Die Entwurfsregeln für Transistoren (Bild 1.11.) haben deshalb die komplexeste Struktur und sind besonders sorgfältig zu prüfen.

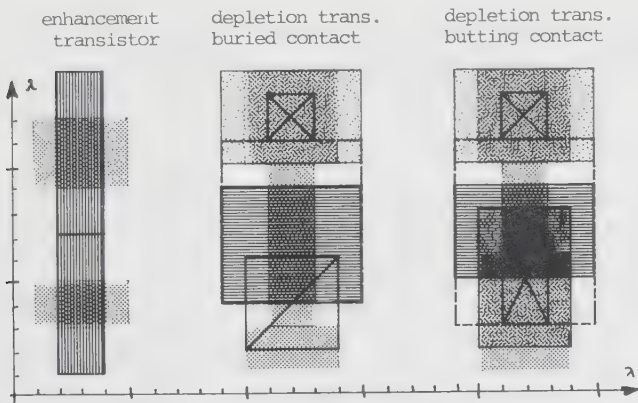


Bild 1.11.: Entwurfsregeln für Transistoren

Geometrische Layoutprüfung

Wurde das Layout wie bei der Fallstudie manuell erstellt, ist eine Überprüfung der geometrischen Entwurfsregeln unumgänglich, während bei symbolischem Layout mit anschließender Kompaktierung die Entwurfsregeln durch das System berücksichtigt werden. Zur Überprüfung der Entwurfsregeln wird eine Expansion der Zellen des Entwurfs vorgenommen. Die

primitiven Elemente dieser ebenen Repräsentation werden dann nach Koordinaten sortiert und zeilenweise untersucht (ZINK 84). Da bei komplexen Entwürfen eine Vielzahl von Abständen und Überlappungen zu prüfen ist, wird der Rechenzeitaufwand sehr hoch. Eine erhebliche Reduktion der Rechenzeit kann durch hierarchische Prüfalgorithmen erreicht werden, bei denen die Topologie der Schaltung und ihre Modularisierung in Zellen bei der Analyse berücksichtigt wird. Zellen die bereits geprüft wurden, werden bei wiederholtem Auftreten nicht wieder vollständig analysiert. Lediglich die Randzone und die Anschlüsse der Zellen werden bezüglich der Einhaltung von Entwurfsregeln untersucht. Werden die Spezifikation der Entwurfsregeln und das Layout primitiver Elemente dem Prüfprogramm als Parameter eingegeben, wird eine weitgehende Unabhängigkeit des Prüfprogrammes von der verwendeten Technologie erreicht. Die Meldung von Verletzungen der Entwurfsregeln erfolgt entweder durch Angabe der Koordinaten oder graphisch, indem an fehlerhaften Stellen zusätzliche Symbole gezeichnet werden.

Elektrische Entwurfsregeln

Elektrische Entwurfsregeln sind für die Funktionsfähigkeit des Schaltkreises notwendige Bedingungen, die auf einfache Weise eine Berechnung der Treiberleistung von Transistoren ermöglichen, ohne eine vollständige elektrische Simulation durchführen zu müssen. Elektrische Entwurfsregeln werden auf der Basis geometrischer Abmessungen der Kanallänge L (Abstand der Diffusionsgebiete von drain und source) und der Kanalbreite W (Breite des Diffusionsgebietes am Kanal) definiert (ANCEAU 81). Bei den Leiterbahnen wird lediglich deren Länge und die Maskenebene in Betracht gezogen. Den Berechnungen liegen die physikalischen Parameter des verwendeten NMOS-Prozesses zugrunde (Bezeichnungen MEAD 80).

Bei den angegebenen Berechnungsregeln ist darauf zu achten, daß stets die effektive Kanalbreite W_{pd} angegeben ist, die ein, dem betrachteten Gatter im elektrischen Verhalten gleicher, Inverter aufweisen würde. Bei einem NOR-Gatter ist die effektive Kanalbreite also gleich der Summe der nominalen Kanalbreiten aller Anreicherungstransistoren des Gatters.

Regeln für das Transistor-Layout

Formverhältnis k : $2,25 \leq (W_{pu}/L_{pu}) / (W_{pd}/L_{pd}) \leq 4$

Dimensionierung Grundgatter $W_{pu} = 2$ $L_{pu} = 3$
 $W_{pd} = 3$ $L_{pd} = 2$

Transfer - Gatter $W_t = 2$ $L_t = 2$

optimaler Treiberfaktor $W_{pd(i)} / W_{pd(i-1)} = 4$
 für Signalverstärkung

Die Ausgangsleistung eines Gatters oder Inverters wird durch die im Signalverlauf zu treibenden Leitungen, Transfergatter und die nachgeordneten Gatter bestimmt. Will man die notwendige Treiberleistung berechnen, geht man also von den Kanalbreiten der angesteuerten Gatter, den zwischengeschalteten Transfergattern und den Längen der Verbindungsleitungen aus und berechnet die effektive Kanalbreite W_{pd} des Anreicherungstransistors des betrachteten Gatters. Die nominalen Kanalbreiten werden dann abhängig vom Gattertyp, die Dimensionierung des Verarmungstransistors W_{pu} durch das Formverhältnis k bestimmt.

Die Schaltzeiten der Gatter bleiben durch die elektrische Anpassung in etwa konstant und ermöglichen somit eine Abschätzung des zeitlichen Verhaltens des gesamten Schaltkreises ohne eine elektrische Simulation.

Berechnungsregel für Anreicherungstransistoren

Effektive Kanalbreite W_{pd} des berechneten Anreicherungstransistors

$W_{pd} = 1/4$ (Summe nominaler, angesteuerter W_{pd} 's)
 + $1/4$ (Summe W_t 's zwischengeschalteter Transfergatter)
 + $1/20$ (Länge Diffusions-Leitungen)
 + $1/40$ (Länge Polysilizium/Aluminium Leitungen)

Prüfung elektrischer Entwurfsregeln und Parameterextraktion

Die Überprüfung der geometrischen Entwurfsregeln wird ergänzt durch eine Analyse des Layouts bezüglich der Transistorgeometrien, der Verbindungen und der elektrischen Entwurfsregeln. Der erste Schritt ist dabei die Erkennung von Transistorstrukturen und ein Aufbau einer Verbindungsliste, die eine Untersuchung der Verbindungsstrukturen erlaubt. Dabei wird geprüft, ob offensichtlich falsche Verbindungen wie Kurzschlüsse oder der Anschluß eines Treibertransistors an die Masseleitung existieren. Wurde vor der Layouterstellung eine äquivalente Transistorschaltung eingegeben, kann durch einen Vergleich des aus dem Layout extrahierten Netzwerkes mit dem Transistor-Schaltbild eine Validierung des Layouts vorgenommen werden.

Eine Überprüfung elektrischer Entwurfsregeln aufgrund der aus dem Layout berechneten Leitungslängen und der Dimensionierung der Transistoren ermöglicht eine statische Prüfung der prinzipiellen Funktionsfähigkeit des Schaltkreises. Da jedoch sowohl die Transistor-Dimensionierung als auch die Berechnungsregeln technologieabhängig sind und auch bei kleineren Änderungen des Prozesses modifiziert werden müßten, gibt es dafür bislang kein Programm, das weitere Verbreitung gefunden hätte. Deshalb wurde bei der Fallstudie die Einhaltung der elektrischen Entwurfsregeln anhand des Layouts überprüft und durch elektrische Simulationen ergänzt (siehe Abschnitt 2.4).

Eine häufig genutzte Form der Schaltungsextraktion ist die Parametergewinnung für die elektrische Simulation. Dabei werden neben der Netzwerkbeschreibung und der Transistor-Dimensionierung auch parasitäre Kapazitäten der Diffusionsgebiete und der Verbindungsleitungen aus dem Layout extrahiert. Die aus dem Layout gewonnenen Daten werden so aufbereitet, daß sie direkt als Eingangsdaten für eine elektrische Simulation verwendet werden können (JERRAYA 83). Dadurch wird es möglich, eine wiederholte Simulation des Schaltkreises durchzuführen und die Auswirkungen von Layout-Optimierungen anhand der Simulationsergebnisse zu überprüfen.

1.2.6. Elektrische Simulation

Eine elektrische Simulation der Zellen eines Entwurfs ermöglicht eine genaue Analyse des elektrischen und zeitlichen Signalverhaltens. Aufgrund der Simulationsergebnisse werden dann die auf der funktionalen Ebene gemachten Angaben über das geforderte zeitliche Verhalten des Entwurfs validiert. Können die zeitlichen Randbedingungen nicht eingehalten werden, ist entweder eine Optimierung des Layouts zur Erreichung höherer Schaltgeschwindigkeiten oder eine Änderung der funktionalen Spezifikation notwendig. Im letzten Fall muß der gesamte Entwurfsprozeß nochmals durchlaufen werden.

Die elektrische Simulation beruht auf technologieabhängigen Modellen der verwendeten Bauelemente. Die Berechnung des Schaltkreisverhaltens erfolgt nach dem Verfahren der Knotenspannungsanalyse (SIBBERT 78), wozu wegen der parasitären Kapazitäten die Lösung eines Differentialgleichungssystems erforderlich ist. Dabei werden iterative Verfahren angewendet, deren Schrittweite abhängig vom geschätzten Fehler und einer vorgegebenen Toleranzgrenze variiert wird.

Die elektrische Simulation bestimmt für den zu analysierenden Schaltkreis das Übergangsverhalten von Spannung und Strom an allen Knoten des Netzwerkes, meist in Abhängigkeit von der Temperatur. Zusätzlich können Sensibilitätsanalysen, Strombedarfsberechnungen und Parameteroptimierungen durchgeführt werden. Eine automatische Variation der Schaltungsparameter, insbesondere der Transistordimensionierung mittels einer vom Anwender zu definierenden Zielfunktion erlaubt eine Schaltungsoptimierung. Wurde eine Zelle unter Berücksichtigung der elektrischen Entwurfsregeln erstellt, kann durch die automatische Optimierung relativ einfach eine Verbesserung des Layouts ermittelt werden.

Anwendung der elektrischen Simulation ist jedoch durch die Anzahl der aktiven Bauelemente begrenzt. Das Differentialgleichungssystem enthält für jeden Knoten der Schaltung eine Gleichung und eine Unbekannte. Die zur Lösung dieses Gleichungssystems notwendige Rechenzeit wächst bei Schaltungen mit mehr als 100 Transistoren so stark, daß eine sinnvolle Simulation nicht mehr möglich ist. Da zu bestimmten Zeitpunkten einer Simulation größere Teile der Schaltung inaktiv oder ohne Einfluß auf das Ablaufgeschehen bleiben, kann für größere Schaltkreise nur eine logische Simulation durchgeführt werden. Dabei werden einfachere Modelle mit

diskreten Werten für die Signalzustände zugrunde gelegt, die eine wesentlich beschleunigte Berechnung erlauben. Neuere Ansätze kombinieren die logische, die zeitliche und die elektrische Simulation (mixed mode simulation), indem nur für kritische Teile des Entwurfs eine Transientenanalyse durchgeführt wird, der Rest der Schaltung jedoch auf der logischen Ebene mit diskreten Werten für die Zustände simuliert wird (DEMAN 82).

1.2.7. Generierung von Testmustern

Bereits beim Entwurf eines Schaltkreises ist die Testbarkeit der Schaltung nach der Fertigung zu berücksichtigen. Bei zunehmender Komplexität integrierter Schaltkreise hat sich die Zahl der nach außen gehenden Anschlüsse kaum erhöht, wodurch immer weniger interne Signale auch von außen meßbar wurden. Konnten bisher beim Test von Leiterplatten Signale zwischen den einzelnen Funktionsblöcken abgegriffen werden, ist der Zugriff bei integrierten Schaltkreisen auf die externen Signale beschränkt.

Testmethoden

Ein vollständiger Test durch Anlegen aller möglichen Kombinationen von Testmustern kann bei hochintegrierten Schaltkreisen nicht durchgeführt werden. Strukturelle Tests, bei denen die Testmuster aufgrund eines angenommenen Fehlermodells generiert werden, ermöglichen eine Reduktion des Testaufwandes, da nur solche Eingaben berücksichtigt werden, die einen mit den vorangegangenen Tests nicht entdeckten möglichen Fehler überprüfen. Für komplexe Schaltungen führt eine Testmustergenerierung mit ausreichender Fehlerüberdeckung (Anzahl der beobachtbaren Fehler) aufgrund der hohen Anzahl der internen Zustände jedoch zu langen Testsequenzen.

Fehlersimulation

Die Fehlersimulation ist eine Erweiterung der logischen Simulation zur Bewertung von Testmustern und der Ermittlung der bei Durchführung des Tests erreichten Testüberdeckung. Dabei wird von einer Beschreibung der

zu testenden Schaltung auf der Gatterebene und definierten Anfangszuständen der Schaltung ausgegangen. Als Fehlermodell wird angenommen, daß ein Signal unabhängig vom Zustand seiner Umgebung ständig den gleichen Wert annimmt ('stuck at'-Fehler).

Aus der Beschreibung des Schaltkreises wird ein Simulator generiert, der sowohl das spezifizierte als auch fehlerhaftes Verhalten zu simulieren gestattet. Werden als Testeingabe die Testmuster verwendet, die auch bei der funktionalen Simulation Anwendung fanden, kann der Fehlersimulator durch einen Vergleich der korrekten Schaltung mit der, die sich bei einem Fehler ergeben würde, feststellen, ob das Testmuster zu einer fehlerhaften Funktion führt. Bei einer Abweichung beider Ergebnisse stellt die Eingabe einen Test für den Fehler dar, der die Abweichung verursacht hat. Der Fehler wird aus der Fehlerliste gestrichen, das Testmuster gespeichert. Ist keine Abweichung der beiden Modelle zu beobachten, ist das Testmuster für die Fehlererkennung irrelevant. Wurden alle Tests angewendet, gibt die Anzahl der restlichen nicht getesteten Fehler Auskunft über die erreichte Fehlerüberdeckung. Die wesentliche Anwendung der Fehlersimulation ist eine Minimierung der Anzahl der Testmuster, die für eine geforderte Testüberdeckung notwendig sind.

Entwurf testfreundlicher Schaltungen

Um die Testbarkeit eines Entwurfs zu verbessern, ist es notwendig, schwer testbare, dynamische Fehler durch eine entsprechende Entwurfsmethode auszuschließen und die Beeinflussung der internen Zustände durch von außen angelegte Signale sowie die Beobachtbarkeit interner Abläufe zu ermöglichen.

Die Beschränkung auf digitale, synchrone Schaltwerke in der hier behandelten Entwurfsmethode erlaubt die Vernachlässigung dynamischer Effekte, wie das Auftreten kurzzeitiger Signalspitzen ('hazards'), da Zustandsänderungen des Schaltwerkes nur zu gewissen Taktzeitpunkten vorkommen können. Asynchrone Rückkopplungen werden durch zwischengeschaltete Register aufgelöst, wodurch Wettlaufbedingungen ('races') nicht auftreten können.

Die Beobachtbarkeit interner Signalzustände wird durch die Verwendung

von LSSD oder BILBO-Elementen statt konventioneller Register ermöglicht (WILLIAMS 83). Dabei werden Register durch zusätzliche Gatter so erweitert, daß sie die Funktion eines Schieberegisters ausüben können. Durch die Einführung eines Schiebetaktes, der von außen steuerbar ist, können alle internen Zustände der Schaltung gesetzt und abgerufen werden. Dazu werden alle Schieberegister mit einem bidirektionalen Datenpfad verbunden, so daß diese zu Testzwecken von außen ladbar und abfragbar sind. Diese Technik erlaubt eine hohe Testüberdeckung bei einer vergleichsweise geringen Anzahl von Testmustern. Die durch die zusätzlichen Gatter und Datenleitungen benötigte Chip-Fläche ist relativ gering (10-20%).

1.3. SILICON COMPILER

Die heute angewendeten Entwurfssysteme unterstützen weitgehend die Entwicklung von Gate Array und Zellenschaltungen. Der Entwurf von Kundenschaltschaltungen ist wegen der hohen Entwicklungskosten nur für standardisierte Anwendungen wie Mikroprozessoren oder bei sehr großen Stückzahlen wirtschaftlich. Anwendungsgebiete mit Problemstellungen, welche die Entwicklung von Spezialschaltungen geringer Stückzahl erfordern, sind nur dann zu erschließen, wenn die Entwicklungskosten für Kundenschaltschaltungen durch den Einsatz weitgehend automatischer Synthesysteme gering gehalten werden.

Das umfassendste Werkzeug, der Silicon Compiler ist jedoch noch Forschungsgegenstand (GRAY 79, WERNER 82). Der Silicon Compiler übersetzt eine problemorientierte Spezifikation der Funktion eines Entwurfs in geometrische Objekte zur Fertigung von Masken, die zur Herstellung eines integrierten Schaltkreises dienen. Wesentlicher Bestandteil ist die Transformation einer problemorientierten Beschreibung des zu synthetisierenden Entwurfs in eine Datenstruktur wie etwa eine geometrische Layoutsprache, welche die physikalische Struktur beschreibt.

In den folgenden Abschnitten werden die Struktur und die wesentlichen Merkmale des CERES Synthese-Systems entwickelt, welches einen von der Technologie unabhängigen Silicon Compiler realisiert sowie aus der Literatur bekannte Synthese-Systeme und ihr Verhältnis zum Inhalt dieser Arbeit vorgestellt.

1.3.1. CERES Synthese-System

Der Spezifikation eines zu realisierenden Schaltkreises in einer technologieunabhängigen Rechner-Entwurfssprache anstatt der geometrischen Beschreibung von Transistoren und Verbindungen, kommt die gleiche Bedeutung zu, wie der Einführung höherer Programmiersprachen in der Software Entwicklung. Die automatische Übersetzung eines problemorientierten Programmes in eine maschinenorientierte Form hat dabei wesent-

liche Gemeinsamkeiten mit der automatischen Synthese integrierter Schaltungen aus einer technologieunabhängigen Spezifikation. Konventionelle Compiler für höhere Programmiersprachen ermöglichen eine problemorientierte Beschreibung und erzeugen eine maschinenorientierte Implementierung. Bei einer Modifikation des Programmes mit anschließender Übersetzung erhält man automatisch einen entsprechend geänderten Maschinencode. Werden in der funktionalen Spezifikation eines Entwurfs für integrierte Schaltkreise Änderungen vorgenommen, erzeugt der Silicon Compiler ein modifiziertes Layout mit neuer Platzierung und Verdrahtung der elementaren Zellen.

Die Theorie der formalen Sprachen ist die Grundlage für die Entwicklung höherer Programmiersprachen und ihrer Übersetzer. Eine ähnliche Modellbildung ist auch auf dem Gebiet der Synthese von Schaltkreisen notwendig, wenn die dabei auftretenden komplexen Problemstellungen formalen Methoden zugänglich gemacht werden sollen. Die funktionale Beschreibung eines Entwurfs in einer Rechner-Entwurfssprache weist keine prinzipiellen Unterschiede zur Programmierung in einer höheren Programmiersprache auf. Das zu synthetisierende Objekt - die integrierte Schaltung - ist jedoch mit einer linearen Struktur nicht hinreichend exakt zu beschreiben. Die geometrischen Strukturen und ihre Relationen erfordern einen Formalismus, der es gestattet, strukturelle und funktionale Eigenschaften zu erfassen und darzustellen sowie formale Mechanismen zur Manipulation bereitstellt. Attributierte Graph-Grammatiken erweisen sich, wegen der einfachen Darstellung mehrdimensionaler Strukturen und der Möglichkeit Prädikate und Funktionen durch Attribute zu definieren, als hervorragend geeignet diesen Aspekten gleichermaßen gerecht zu werden.

Für die Definition der Syntax und Semantik von Programmiersprachen werden attributierte Graph-Grammatiken bereits seit mehreren Jahren erfolgreich angewendet (GÖTTLER 77, BRENDDEL 79). Die Definition einer Rechner-Entwurfssprache mit Hilfe der Graph-Grammatiken ist deshalb der Ausgangspunkt für die Spezifikation des Übersetzungsprozesses mit dem gleichen formalen Hilfsmittel. Andererseits sind auf dem Gebiet der Mustererkennung eine Reihe von Arbeiten bekannt, die Graph-Grammatiken zur Beschreibung komplexer geometrischer Strukturen und ihrer Analyse einsetzen (BUNKE 83, BARTSCH 83). Die formale Definition der Struktur hochintegrierter Schaltkreise beinhaltet ähnliche Anforderungen.

Das CERES Synthese-System (siehe Bild 1.12) basiert auf der Anwendung

attributierter Graph-Grammatiken sowohl für die Definition der Rechner-Entwurfssprache, als auch die Beschreibung hochintegrierter Schaltkreise. Diese einheitliche Methodik ermöglicht die formale Spezifikation des Syntheseprozesses als Syntaxanalyse in der Theorie der Graph-Grammatiken.

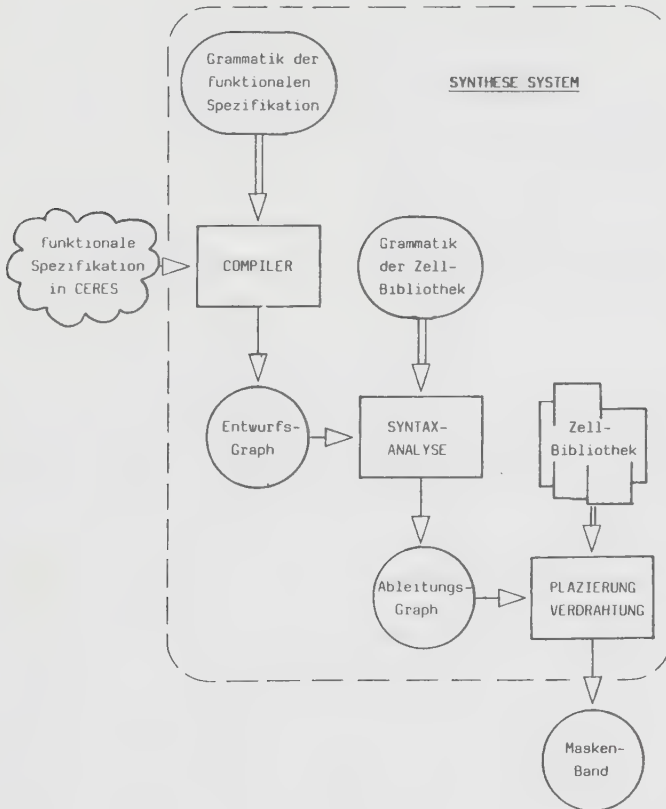


Bild 1.12: Struktur des Synthese-Systems

Ausgehend von einer funktionalen Beschreibung des Entwurfs und seiner Simulation auf der Register-Transferebene erfolgt eine Übersetzung in die interne Repräsentation als Entwurfsgraph. Dieser enthält die gesamte Information der funktionalen Spezifikation und dient zugleich als Ausgangspunkt der Syntexanalyse. Die Vorgehensweise bei der Übersetzung ist ähnlich wie bei Programmiersprachen. Ausgehend von einer formalen Definition der Entwurfssprache wird der Entwurfsgraph durch sukzessive Anwendung von Graph-Ersetzungen konstruiert. Optimierungen des Entwurfs werden als Transformationen dieses Entwurfsgraphen beschrieben. Die primitiven Elemente des Entwurfsgraphen entsprechen dabei den Operationen der Entwurfssprache. Da diesen während des Syntheseprozesses Objekte der Technologie zugeordnet werden müssen, ist es notwendig die Sprachkonstrukte so zu wählen, daß dafür eine technologische Realisierung in der Zellbibliothek existiert. Umgekehrt besteht aber auch die Notwendigkeit, typische Elemente integrierter Schaltkreise in der funktionalen Beschreibung spezifizieren zu können.

Die elementaren Bausteine eines integrierten Schaltkreises sind Zellen, deren Layout interaktiv entworfen wurde und deren Funktion und Leistungsmerkmale durch Simulation auf der Transistorebene validiert wurden. Um diese technologische Realisierung elementarer Funktionen für eine Synthese nutzbar zu machen, ist es notwendig, ihre Eigenschaften abstrakt zu modellieren. Die Repräsentation einer Zelle durch einen attribuierten Graphen erlaubt die Modellierung struktureller Eigenschaften wie die Topologie der Anschlußpunkte durch Knoten und Kanten, während zeitliche Eigenschaften durch Attribute beschrieben werden (vergleiche Beispiel 4.2). Wählt man elementare logische Operationen als Knotenmarkierung, enthält der Zellgraph keine technologieabhängigen Strukturen. Zum abstrakten Modell einer Zelle können durchaus mehrere Realisierungen in verschiedenen Technologien existieren, wenn die logische Funktion der Knoten entsprechend implementiert wird. Diese technologieunabhängigkeit der formalen Beschreibung ist eine notwendige Voraussetzung für eine breite Anwendung des Synthese-Systems. Da die Zellen einer Zellbibliothek im allgemeinen nicht beliebig untereinander kombinierbar sind, muß darüber hinaus eine Vorschrift erstellt werden, wie bestimmte Zellen untereinander verdrahtbar sind. Diese Vorschrift kann durch eine attribuierte Graph-Grammatik formal erfaßt werden, wobei die Attribute eine Spezifikation der durch die Verbindung von Zellen resultierenden Eigenschaften erlauben (siehe Bild 4.2). In der Sprechweise der Theorie der Graph-Grammatiken definiert die Grammatik der

Zellen die Menge aller mit dieser Zellbibliothek realisierbaren Schaltungen, während die Grammatik der funktionalen Spezifikation alle beschreibbaren Entwürfe erzeugt (siehe Bild 1.13). Die gemeinsame Teilmenge ist dann die Menge aller Entwürfe, die aus der Zellbibliothek synthetisiert werden können.

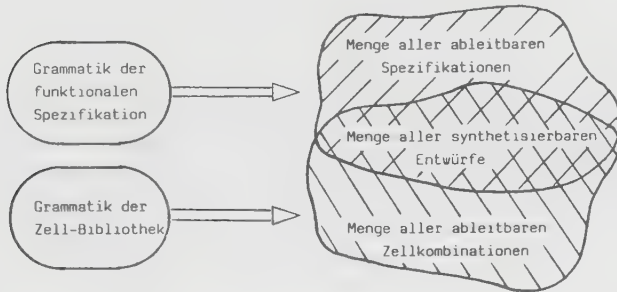


Bild 1.13: Grammatiken und erzeugte Wortmengen

Das Syntheseproblem kann damit auf das Wortproblem in der Theorie der Graph-Grammatiken zurückgeführt werden. Dies bedeutet, daß das Synthesesystem erkennen muß, ob der Entwurfsgraph mit Hilfe der Zellgrammatik aus der Zellbibliothek ableitbar ist. Da der Entwurfsgraph und das abstrakte Modell der Zellen aus den gleichen elementaren Objekten aufgebaut sind, kann dies mit Hilfe einer Syntaxanalyse durchgeführt werden. Die Synthese eines integrierten Schaltkreises kann deshalb als Syntaxanalyse bei attributierten Graph-Grammatiken realisiert werden. Ein Silicon Compiler als Synthesesystem hat damit ähnliche Eigenschaften wie ein konventioneller Compiler für Programmiersprachen. Während der Syntaxanalyse auftretende Fehler, ermöglichen gezielte Hinweise auf notwendige Änderungen in der funktionalen Beschreibung des Entwurfs, ohne daß die sonst übliche Simulation auf der Gatterebene durchgeführt werden muß. Das Ergebnis der Syntaxanalyse ist ein Ableitungsbaum, der angibt, aus welchen Zellen der Bibliothek der Entwurf besteht und wie diese zusammengefügt werden müssen. Er enthält damit alle Informationen, die für eine Realisierung des Entwurfs als integrierte Schaltung notwendig sind und ist die Ausgangsbasis für die

Erstellung des Maskenbandes durch Algorithmen zur Platzierung und Verdrahtung der Zellen. Darüber hinaus ermöglicht das Synthese-System die Durchführung der Synthese eines Entwurfs mit verschiedenen Zellbibliotheken möglicherweise unterschiedlicher Technologie, wodurch auf einfache Weise verschiedene Realisierungsmöglichkeiten erprobt werden können.

1.3.2. Vergleich verschiedener Synthese-Systeme

Die Entwicklung von Systemen zur Synthese digitaler Schaltwerke geht einher mit der Einführung von Register-Transfersprachen zu Beginn der sechziger Jahre (ANDERSON 62, SCHORR 64). Diese frühen Systeme hatten die Transformation von Register-Transfer Beschreibungen in boolesche Gleichungen zum Inhalt. Die Entwicklung hochintegrierter Schaltkreise hat eine wesentliche Verstärkung dieser Aktivitäten hervorgerufen. Die wichtigsten neueren Ansätze zur Synthese und die in ihnen realisierten Synthesetechniken werden im folgenden behandelt.

CMU-DA: Carnegie-Mellon University Design Automation Project

Die Synthese der logischen Schaltungsstruktur aus einer algorithmischen Beschreibung des Entwurfs in der Rechner-Entwurfssprache ISPS steht im Mittelpunkt des Carnegie-Mellon University Design Automation Project (HAFER 82). Das CMU-DA System realisiert eine top down Synthese, ausgehend von der Übersetzung der ISPS Spezifikation in eine interne globale Datenstruktur. Diese dient einerseits der Generierung eines Simulators auf der Register-Transferebene und ist andererseits Datenbasis für den Optimierer, der eine Datenfluß-Repräsentation des Entwurfs generiert. Kern des Systems ist ein Bewertungsmodul, der aufgrund vom Entwickler vorgegebener Optimierungskriterien wie Geschwindigkeit und Kosten eine bestimmte Architektur auswählt (Busarchitektur oder verteilte Logik). Abhängig von der gewählten Architektur werden dann verschiedene Synthese-Moduln aufgerufen, die eine Datenpfadstruktur erzeugen. Verschiedene heuristische Synthesetechniken ergänzen das System, unter anderem ein Modul zur Synthese eines mikroprogrammierten Leitwerks. Das Synthese-System setzt die Verwendung bestimmter Architekturen voraus. Dies ist im wesentlichen auf die algorithmische Spezifikation des Entwurfs zurückzuführen, die keine direkte Entsprechung der

entwurfssprachlichen Konstrukte mit den Elementen hochintegrierter Schaltungen bietet.

MIMOLA Entwurfs-System

Das MIMOLA Entwurfssystem (ZIMMERMANN 79, MARWEDEL 79) basiert auf einer Rechner-Entwurfssprache, welche mikroprogrammierte Rechenwerke und Mikroprogramme zu beschreiben und simulieren gestattet. Ziel ist weniger die automatische Synthese als die Unterstützung des Entwicklers bei der Erprobung verschiedener Entwurfsalternativen. Dazu werden Teile des Entwurfs als Makro deklariert, die dann zu Datenpfad und Leitwerk zusammengesetzt werden. Durch schrittweise Verfeinerung kann der Entwurf von einer algorithmischen Beschreibung auf der Systemebene zu einer funktionalen Spezifikation auf der Register-Transferebene erweitert werden. Dabei wird von einem mikroprogrammierten Leitwerk ausgegangen, weshalb das MIMOLA System für den Entwurf allgemeiner hochintegrierter Schaltungen nur eingeschränkt verwendbar ist.

LOGE Synthese-System

Das LOGE Synthese-System (GRASS 79, LIPP 82) ermöglicht die Synthese von Steuerungen aus einem Funktionsplan. Die zu steuernde Einheit ist dabei nicht unbedingt ein digitales Rechenwerk, sondern kann aus einem technischen Prozess bestehen. Ausgehend von der funktionalen Spezifikation in Form eines Schrittschaltwerkes können verschiedene Realisierungen synthetisiert werden. Das Spektrum der einzelnen Synthesemodule ermöglicht die Generierung von Hardware, Firmware und Software Lösungen mit konventioneller Technik.

Synthese-Systeme für reguläre Strukturen

Für die Herstellung von Gate Arrays werden Systeme eingesetzt, die ausgehend von einem konventionellen Schaltplan automatisch die Platzierung und Verdrahtung vordefinierter Zellen durchführen. Ergänzt man diese Systeme um eine programmiersprachliche Darstellung der Struktur des Entwurfs, kann eine Übersetzung in die Schaltplandarstellung der Gatterebene vorgenommen werden. Das MODEL Entwurfssystem (GRAY 82) geht

von einer strukturellen Beschreibung des Entwurfs aus, und übersetzt diese in eine Netzliste der verwendeten Elementarzellen. Eine Modularisierung des Entwurfs ist durch die Deklaration einer Hierarchie von Zellen und ihrer Verbindungen möglich. Die Übersetzung der Strukturdefinition in ein Netz von Gattern erlaubt dann eine logische Simulation auf der Gatterebene. Die Platzierung und Verdrahtung erfolgt mit den für Gate Arrays verfügbaren Werkzeugen. Da die Entwurfssprache MODEL keine Kontrollstrukturen für eine funktionale Beschreibung des Ablaufgeschehens und keine Taktdeklaration enthält, kann lediglich die Struktur des Entwurfs ähnlich einem Blockschaltbild beschrieben werden.

Ein System zur Synthese synchroner Schaltungen durch die Verwendung von PLA's ('Programmable Logic Array', siehe Abschnitt 4.3.3) beschreibt (AYRES 79). Die Entwurfssprache erlaubt die Spezifikation durch boolesche Gleichungen, die nach ihrer Übersetzung in eine zweistufige Normalform als PLA implementiert werden. Ergänzt man diese reguläre Struktur durch Inverter und Register, kann damit jedes synchrone Schaltwerk realisiert werden. Der erhebliche Platzbedarf einer solchen Lösung kann durch Partitionierung des PLA's in mehrere Module vermindert werden, wozu eine hierarchische Strukturierung des Entwurfs vorgenommen wird. Die Verwendung standardisierter Gatter (LUHUKAY 82) ermöglicht eine kompaktere Realisierung, erfordert jedoch die Anpassung der Transistordimensionen um elektrische Randbedingungen einzuhalten. Im Gegensatz zum MODEL System ist die Entwurfssprache beider Ansätze weniger an der Struktur als an der logischen Funktion des Schaltkreises orientiert, erlaubt jedoch keine Beschreibung des Entwurfs auf der Register-Transferebene. Eine funktionale Simulation ist deshalb nur auf der Gatterebene möglich.

Synthese-Systeme für bestimmte Architekturkonzepte

Beim Synthese-System BRISTLE BLOCKS, für den hierarchischen Entwurf des Datenpfades von Mikroprozessoren (JOHANNSEN 79), wird eine mikroprogrammierte Architektur zugrunde gelegt, deren Kern der Datenpfad mit zwei Datenbussen darstellt. Der gesamte Entwurf wird strukturell durch eine Hierarchie von Zellen beschrieben, die entweder geometrische Elemente oder weitere Zellen enthalten. Der Entwurf der primitiven Zellen erfolgt durch den Entwickler, um ein optimales Layout zu erzielen. Da keine funktionale Spezifikation des Entwurfs erfolgt, ist

keine Simulation möglich. Die Synthese besteht in der automatischen Verdrahtung der Zellen und ihrer Platzierung durch das System.

Die Architektur des Datenpfades steht auch im Mittelpunkt des McPITTS Systems (SISKIND 82). Die Entwurfssprache ist im Gegensatz zu BRISTLE BLOCKS funktionsorientiert, bei dem eine strukturorientierte Spezifikation Ausgangspunkt der Synthese ist. Der McPITTS Compiler besteht aus zwei Phasen. Zuerst wird die funktionale Spezifikation in eine Reihe technologieunabhängiger Datenpfad-Spezifikationen, Zustandsgleichungen und Kontrollanweisungen übersetzt. In einem zweiten Schritt werden aus diesen Gleichungen der Datenpfad und die Steuerlogik erzeugt. Die Entwurfssprache ist an der Syntax von LISP orientiert und gestattet eine nichtprozedurale Beschreibung der Funktion. Kontrollstrukturen für die bedingte Ausführung von Register-Transfers und die Beschreibung von Nebenläufigkeiten sind in der Entwurfssprache enthalten, jedoch keine Zeitdeklarationen.

Der experimentelle Silicon Compiler DEA (DEsign Architecture, RUPP 81) verwendet als Entwurfssprache eine Teilmenge der Programmiersprache C, die um einen binären Datentyp erweitert wurde. Nicht enthalten sind Kontrollstrukturen zur Beschreibung des Ablaufgeschehens. Elementare Operatoren sind primitive Elemente der NMOS-Technologie, wie Transistoren und Schalter. Das Zeitmodell geht von einem verzögerungsfreien Schalten kombinatorischer Netzwerke aus, während für speichernde Elemente ein Zeittakt als Verzögerung angenommen wird. Die Simulation ermöglicht deshalb lediglich eine funktionale Validierung, jedoch keine Leistungsbewertung. Die Synthese einer integrierten Schaltung erfolgt mittels einer Ersetzung der Operatoren durch Elementarzellen der verwendeten Technologie, deren topologische Verbindung vom System erzeugt wird. Dabei tritt das Problem der Generierung von Anschlußpunkten für die Verdrahtung der Zellen auf ('side assignment' Problem). Da die Synthese der Topologie bottom up erfolgt, werden die Anschlußpunkte so gelegt, daß der Flächenbedarf im Inneren der Zelle minimiert wird. Dies führt jedoch nicht unbedingt zu günstigen Lösungen auf der nächst höheren Ebene der Zellhierarchie.

Allen bisher veröffentlichten Synthese-Systemen gemeinsam sind heuristische Algorithmen zur Transformation algorithmischer, funktionaler oder struktureller Beschreibungen in bestimmte Implementierungstechniken. Die

Verwendung traditioneller Rechner-Entwurfssprachen, deren elementare Objekte und Kontrollstrukturen unzureichend auf elementare Konstrukte hochintegrierter Schaltkreise abgestimmt sind, erschwert dabei die Synthese.

*Good judgement comes from experience -
experience comes from bad judgement.*

Lipman

2. VLSI – FALLSTUDIE BUBBLE-SORT BAUSTEIN

Im Rahmen des von IMAG Grenoble organisierten Multi-Project-Chips CMP'83 (GUYOT 83) wurde mit den in Erlangen verfügbaren Entwurfshilfsmitteln ERES, SPICE und LUCIE ein hochintegrierter Schaltkreis in NMOS Technik entworfen. Ziel des Vorhabens war die Erprobung des Übergangs von einer funktionalen Spezifikation in ERES zum geometrischen Layout des Schaltkreises. Dabei sollten insbesondere die in ERES verfügbaren Sprachmittel auf ihre Anwendbarkeit zum Entwurf hochintegrierter Schaltkreise und Korrespondenzen zwischen funktionaler Beschreibung und Topologie des Layouts untersucht werden. Die dabei gewonnenen Erfahrungen bilden die Grundlage für die in den folgenden Kapiteln beschriebene VLSI-Entwurfssprache und die Synthese Methoden.

2.1. SORTIERALGORITHMEN FÜR VLSI

Die Mikroelektronik ermöglicht die kostengünstige Realisierung unkonventioneller Rechnerarchitekturen unter Ausnutzung inhärenter Parallelitäten. Sortieralgorithmen gehören zur Klasse der Algorithmen, für die exakte Abschätzungen ihrer Speicher- und Zeitkomplexität bekannt sind. Diese Aussagen treffen jedoch für parallele Realisierungen nicht mehr zu. So hat der BUBBLE SORT Algorithmus (WIRTH 76) in der sequentiellen Variante eine höhere Zeitkomplexität als vergleichbare Sortieralgorithmen. Die Lokalität und teilweise Unabhängigkeit der beim Sortiervorgang auftretenden Vertauschungsoperationen erlauben jedoch eine einfache Parallelisierung. Da bei hochintegrierten Schaltkreisen die Länge der Verbindungsleitungen die Schaltgeschwindigkeit wesentlich beeinflusst, ist die Lokalität von Operationen ein entscheidendes Kriterium für die Realisierbarkeit eines parallelen Algorithmus in Form eines integrierter Schaltkreises.

Vorschläge zur Realisierung von Sortieralgorithmen in VLSI wurden in den

letzten Jahren mehrfach veröffentlicht. Sie basieren auf parallelen Versionen des 'BUBBLE SORT' Algorithmus und gestatten das Sortieren von Zahlen in linearer Zeit. Die für den SORT Baustein gewählte Implementierung ist ähnlich der in (CAREY 82) beschriebenen Variante, mit dem Unterschied, daß auf eine Überlagerung von Lade- und Sortierphase verzichtet wurde. Dadurch ist es möglich, den Baustein in der Funktionsweise eines Kellers zu betreiben (GUIBAS 82), was den Test des gefertigten Chips wesentlich vereinfacht.

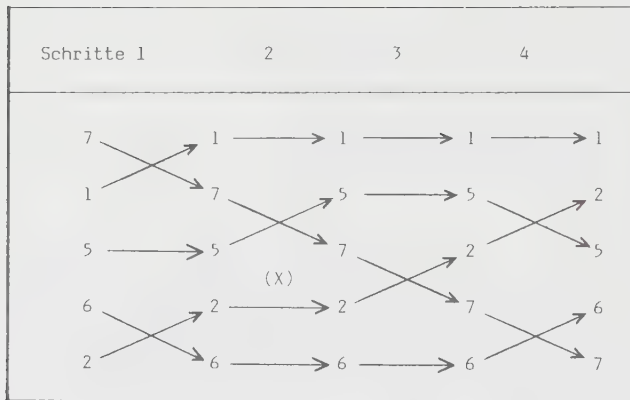
Sortierbausteine sind weniger als Ergänzung konventioneller Prozessoren zur Beschleunigung von Sortiervorgängen geeignet, da Lade- und Leseoperationen einen erheblichen Anteil der Gesamtzeit beanspruchen. Anwendungsgebiete sind vielmehr Spezialprozessoren, die einen hohen Anteil von Sortieraufgaben bewältigen müssen. So ist es bei der Implementierung relationaler Datenbanken notwendig, Zugriffe auf Externspeicher abhängig von sortierten Schlüsseln zu organisieren. Eine Steuerung des Externspeichers, die es ermöglicht Datenbereiche zwischenzuspeichern und sortiert weiterzuleiten, würde indexsequentielle Zugriffe wesentlich beschleunigen (ULLMAN 80). Ebenso wäre der Einsatz in Verschlüsselungsautomaten zur Codierung von Sprache in Echtzeit ein Anwendungsgebiet, welches das schnelle Sortieren von Informationen erfordert (BETH 83).

2.2. FUNKTIONALE SPEZIFIKATION UND SIMULATION

Der BUBBLE SORT Algorithmus sortiert N Zahlen in aufsteigender Reihenfolge. Das Sortierprinzip ist denkbar einfach:

- 1) Notiere N Zahlen in beliebiger Reihenfolge untereinander.
- 2) Vergleiche je zwei direkt benachbarte Zahlen und vertausche beide, wenn die größere über der kleineren steht. Die Vertauschung unterbleibt, wenn die obere Zahl aufgrund des nächst höheren Vergleichs bereits zur Vertauschung vorge-merkt wurde (Konfliktfall).
- 3) Sind alle Zahlenpaare so angeordnet, daß stets die kleinere Zahl oben steht, liegen die Zahlen in aufsteigender Folge vor. Sonst wiederhole ab Schritt 2.

Die parallele Implementierung dieses Algorithmus nutzt aus, daß bei den Vergleichs- und Vertauschungsoperationen keine Aussage über die Reihenfolge, in der diese vorzunehmen sind, gemacht wird. Man kann also parallel Zahlenpaare vergleichen und vertauschen. Ein Konflikt tritt auf, wenn drei oder mehr Zahlen in umgekehrter Sortierreihenfolge stehen und direkt benachbarte Zahlenpaare in einem Schritt vertauscht werden sollen (siehe Schritt 2 in Beispiel 2.1). Man kann dies dadurch umgehen, daß in einem parallelen Schritt stets nur Zahlenpaare verglichen werden, die keine Zahl gemeinsam haben. Dann ist jedoch die doppelte Anzahl von Sortierschritten erforderlich. In der hier beschriebenen Realisierung werden in jedem Schritt alle Zahlenpaare verglichen. Bei einem auftretenden Konflikt wird dann eine der Vertauschungsoperationen um einen Schritt verzögert. Dies liefert im allgemeinen eine schnellere Sortierung.



Beispiel 2.1: Paralleles BUBBLE-SORT Schema mit Konfliktfall (X)

2.2.1. ERES Beschreibung des SORT Bausteins

Eine naheliegende Implementierung des BUBBLE SORT Algorithmus als integrierter Schaltkreis ist, die zu sortierenden Zahlen in Registern zu speichern und jeweils zwischen zwei Register eine Operationseinheit zum Vergleich der Registerinhalte und deren Vertauschung zu plazieren. Zum Laden der Daten und Entladen der sortierten Reihenfolge wird die gesamte Schaltung als Kellerspeicher betrieben, indem durch externe Steuersignale der Datentransfer zwischen den Registern unabhängig vom Ergebnis der Vergleichsoperation beeinflusst wird. Beim Test der gefertigten Schaltung ermöglicht diese Betriebsart eine einfache Überprüfung der Funktionsfähigkeit der Register unabhängig von der Funktionsfähigkeit der Operationseinheiten.

Der parallele BUBBLE SORT Algorithmus hat drei Operationsphasen. Während der Ladephase werden die Daten über die externe Schnittstelle geladen, dann in der anschließenden Sortierphase sortiert. Diese Sortierung erfolgt jeweils in zwei Teilschritten. Im ersten Schritt werden die Lade- und Vertauschungsoperationen vorgenommen, während im zweiten die Vergleichsoperation und die Konfliktbehandlung erfolgen. Diese zwei Teilschritte werden mit den beiden Takten PHI1 und PHI2 der Uhr

gesteuert. Das Ende des Sortiervorganges liegt vor, wenn keine der Operationseinheiten eine Vertauschung der Registerinhalte anfordert. In der Entladephase können die sortierten Daten dann entkellert werden.

Um beim Test des Schaltkreises die Arbeitsgeschwindigkeit extern beeinflussen zu können, werden die Taktsignale (PHI1, PHI2) und die Steuersignale (CHIPSEL, READ) von außen zugeführt. Die Selektion des Bausteines (CHIPSEL=1) erzwingt unabhängig vom internen Zustand die Betriebsart eines Kellerspeichers, während Lade- und Entladephase durch das Steuersignal READ unterschieden werden. Dies ermöglicht beim Test eine Unterbrechung des Sortiervorganges nach jedem Sortierschritt und somit eine leichte Überprüfung der Funktion der Operationseinheiten. Die internen Steuersignale werden aus den externen in einer globalen Steuerlogik erzeugt.

Die funktionale Spezifikation des SORT Bausteins in ERES (GARDILL 77) beschreibt die externen Schnittstellen durch die Register DATE, CHIPSEL und READ ohne Zeitverzögerung, die während der Simulation mit entsprechenden Werten versorgt werden, da ERES keine modulare Beschreibung mit entsprechenden Schnittstellenparametern enthält. Die internen Steuersignale POPN und PUSHN werden aus dem externen CHIPSEL und READ abgeleitet und erzwingen die Datentransfers während der Lade- und Entladephasen.

design BUBBLE_SORT

```

register :0:  DATA[7:0], CHIPSEL,  READ;
init      CHIPSEL=0(1),  READ=0(1);
clock :320,180: PHI[1:2];
terminal :60:  POPN   = CHIPSEL nand READ,
               PUSHN  = CHIPSEL nand not READ,
               READY  = PUSHN and nor SWAPRQ[;8];

```

Der Datenpfad des SORT Bausteins besteht aus acht Registern mit sieben dazwischen angeordneten Vergleichs- und Vertauschungselementen. Da die zur Realisierung des Chips zur Verfügung stehende Fläche begrenzt war, wurden acht Bit als Breite des Datenpfades gewählt. Die reguläre Struktur von acht Registern mit je acht Bit und zugeordneten Verglei-

chern spiegelt sich in der funktionalen Beschreibung durch die Verwendung des Strukturoperators 'forall ... is' wieder, der in ERES jedoch nicht implementiert ist. Er beschreibt in kompakter Form eine Menge von Deklarationen, die bei sukzessiver Ersetzung der Indizes durch die Zahlen des Laufbereichs entstehen. Dieses Sprachelement der ERES-Erweiterung (siehe Abschnitt 4.3.2) wurde aus Gründen der Übersichtlichkeit verwendet.

```
forall I = (1..7) is
  forall J = (8..1) is
    register :10:  R[I;J], R[8;J];
    terminal :20:  SWAPRQ[I;J]=
                  ( R[I;J]nand not R[I+1;J]) nand
                  (SWAPRQ[I;J-1]nand (not R[I;J]nand R[I+1;J])),
                  SWAPRQ[I;0]= 0;
  endall J;
endall I;
```

Die Vergleiche erzeugen bei einer Verletzung der Reihenfolgebedingung das Signal SWAPRQ, welches das Vertauschen der beiden Registerinhalte anfordert. Da dies nur während der Sortierphase und nicht beim Laden und Entladen vorgenommen werden soll, muß jedem Vergleich eine lokale Steuerung zugeordnet werden, die den Datenaustausch in diesen Phasen durch Verknüpfung mit POPN und PUSHN verhindert.

```
forall I = (1..7) is

terminal :10:  SWAP[I]= if not SWAP[I-1]then SWAPRQ[I;8]else 0,
               SWAP[0]= 0,
               :20:  UPIN[I]= if PUSHN then
                       (POPN nand (POPN nand SWAP[I])) else 0,
               :20:  DOWNIN[I]= if POPN then
                       (PUSHN nand (PUSHN nand SWAP[I])) else 0;
register :20:  UP[I], DOWN[I];
endall I;
```

Die Steuerung des gesamten SORT Bausteins erfolgt durch getaktete Register-Transfers, die den Datenfluß zwischen Registern abhängig von Aktivierungsbedingungen beschreiben. Die erste Taktphase PHI1 aktiviert den Transfer zwischen den Datenregistern.

```
forall I = (1..7) is

    / PHI[1]and not PUSHN /      R[1]    <- DATA;
    / PHI[1]and DOWN[I]/        R[I+1]  <- R[I];

    / PHI[1]and UP[I]/          R[I]    <- R[I+1];
    / PHI[1]and not POPN /      DATA    <- R[1];

    / PHI[2]/                    UP[I]   <- UPIN[I],
                                DOWN[I]  <- DOWNIN[I];

endall I;

end; BUBBLE_SORT
```

Während der zweiten Taktphase PHI2 werden die Vergleichsoperationen ausgeführt und die Auswertung der Vertauschungsanforderungen in der lokalen Steuerlogik vorgenommen. Die Steuersignale für den nächsten Bearbeitungsschritt werden in die Steuerregister UP und DOWN übernommen.

Die Grundstruktur der lokalen Steuerung mit zwei Steuerregistern UP und DOWN sowie der zugehörigen Logik ist symmetrisch. Diese Symmetrie der logischen Funktion wird sich im Layout der Zelle durch die symmetrische Verwendung der gleichen Elementarzellen wieder finden.

Die Grundstruktur der Schaltung für eine Zeile des SORT Bausteins mit zwei Registern, dem dazwischen liegenden Vergleich, der lokalen Steuerung und den Passtransistoren zum Schalten der Datenpfade zeigt Bild 2.1. Diese Grundstruktur wiederholt sich in vertikaler Richtung und bildet das gesamte Sortierfeld.

Der Grundriß des gesamten Bausteins kann aufgrund der regulären Struktur der funktionalen Beschreibung und einer ersten Abschätzung des Flächenbedarfs der einzelnen Zellen bereits jetzt erstellt werden (Bild 2.2).

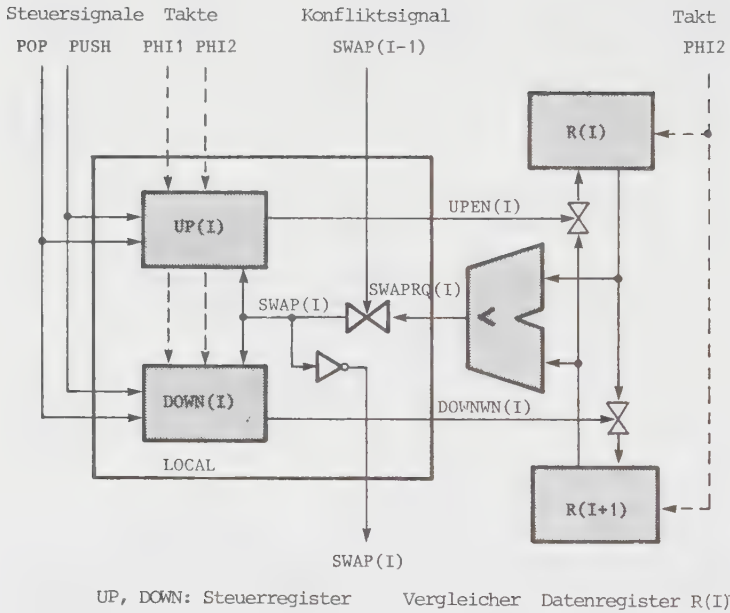


Bild 2.1: Steuer- und Datenregister und Vergleicher

Dieser Grundriß gibt für den Entwurf der einzelnen Zellen wichtige Hinweise, da benachbarte Zellen möglichst so entworfen werden, daß keine gesonderte Verdrahtung zwischen den Zellen notwendig wird. Vielmehr werden die Leitungen der externen Signale so am Zellenrand angelegt, daß bei direkt benachbarten Zellen eine Verbindung durch direktes Anfügen der Zellen entsteht (vergleiche Abschnitt 1.2.4 und ANCEAU 82).

Abkürzungen:

REG8

Datenregister

COM8

Vergleicher

LOCAL

lokale Steuerung

GLOBAL

globale Steuerung

ZIN

Eingangsstufe

ZOUT

Ausgangsstufe

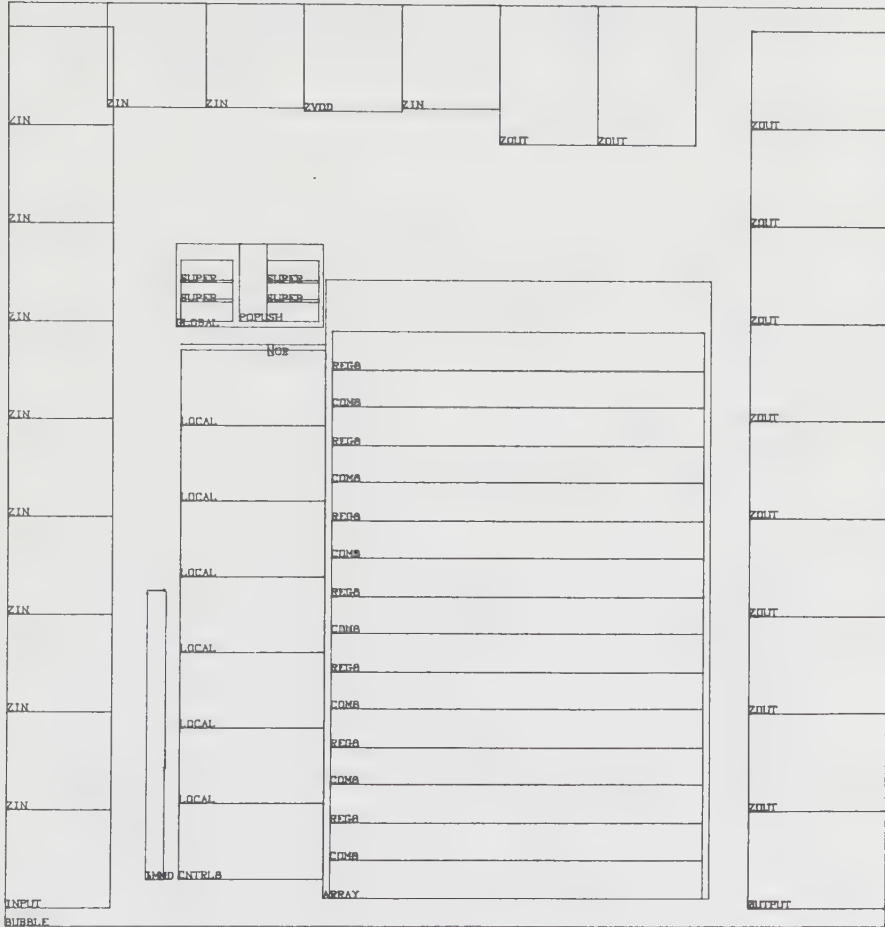


Bild 2.2: Grundriß des SORT Bausteins

2.2.2. Simulation und Leistungsbewertung

Eine Überprüfung des Entwurfs durch eine funktionale Simulation auf der Register-Transferebene ist bei der Fertigung hochintegrierter Schaltkreise ein wesentliches Hilfsmittel zur Validierung und Leistungsbewertung. Die üblicherweise durchgeführte Logiksimulation setzt die Übertragung der funktionalen Beschreibung auf die Gatterebene voraus, was eine iterative Verbesserung des Entwurfs wesentlich erschwert.

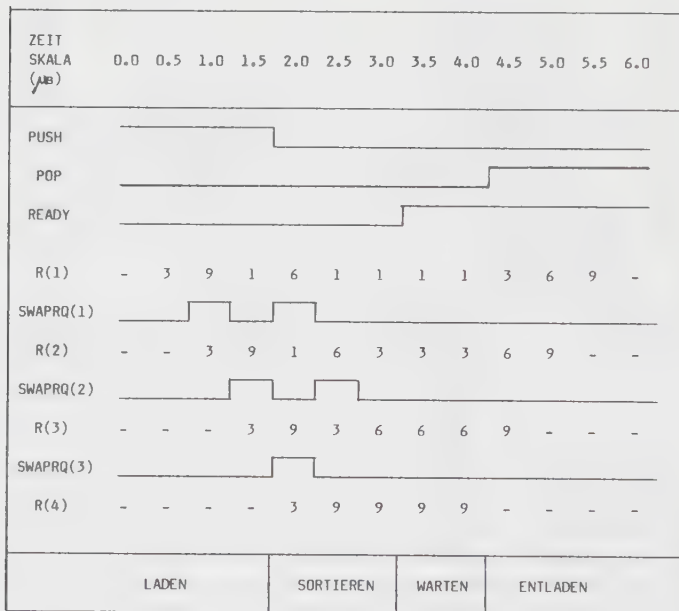


Bild 2.3: Simulationsspur für die Sortierung (vier Register)

Die Spezifikation des Entwurfs in ERES ermöglichte eine funktionale Simulation des Bausteins über eine strukturelle und nicht algorithmisch

angelegte Bausteinbeschreibung. Diese Strukturbeschreibung war in dem Sinne vollständig, daß die replikativen Strukturen in expliziter Form notiert wurden, was eine umfangreichere Beschreibung zur Folge hatte. Die in Bild 2.3. dargestellten Simulationsergebnisse wurden direkt aus dem ERES Simulationsprotokoll übernommen und verdeutlichen die Funktionsweise des SORT Bausteins.

Die Simulation läßt deutlich mehrere Phasen des Sortiervorganges erkennen. Zuerst werden die zu sortierenden Werte gekellert, dann sortiert und zuletzt wieder entkellert. Bemerkenswert dabei ist, daß das Ende des Sortiervorganges genau dann gemeldet wird, wenn keine Anforderung zum Vertauschen der Registerinhalte mehr vorliegt. Dies ist im allgemeinen nach weniger als N Schritten der Fall. Lediglich wenn die kleinste zu sortierende Zahl zuerst oder die größte zuletzt gekellert werden, sind N Sortierschritte durchzuführen, da dann beim Sortieren in eine aufsteigende Folge eine Zahl durch das gesamte Registerfeld wandern muß.

Nach dem Ende des Sortiervorganges werden die Daten entkellert. Da der SORT Baustein in den verschiedensten Anwendungsgebieten Verwendung finden kann, ist es zulässig, daß zwischen der Fertigmeldung des SORT Bausteines ($READY=1$) und der externen Anforderung durch CHIPSEL und READ zum Entkellern der Daten ($POP=1$) mehrere Zyklen verstreichen. Der Baustein ist während dieser Phase inaktiv.

Bei genauer Betrachtung der Simulation bemerkt man, daß bereits während des Kellerns der Daten die Vergleichsoperatoren Reihenfolgeverletzungen feststellen und entsprechende Signale erzeugen. In der vorliegenden Implementierung werden diese in den lokalen Steuereinheiten unterdrückt. Durch eine Erweiterung der Steuerlogik und einen zusätzlichen Datenpfad, der das Überspringen eines Registers erlaubt, kann eine Sortierung bereits während der Ladephase erfolgen. Dabei werden kleine Zahlen verzögert, so daß größere Zahlen diese beim Kellern überholen können. Analog werden beim Entkellern dann größere Zahlen verzögert, so daß eine getrennte Sortierphase entfallen kann. Anschaulich interpretiert bedeutet dies, daß größere Zahlen im Registerfeld beim Kellern schneller sinken und beim Entkellern langsamer aufsteigen als kleine.

Dieser Algorithmus wurde in (CAREY 82) realisiert. Die zusätzliche Logik und die längeren Leitungen verlangsamten jedoch die Taktgeschwindigkeit des gesamten Schaltkreises auf etwa die Hälfte. Der Flächenbedarf für

das Sortierfeld mit Steuerlogik ist um etwa 45% größer als bei diesem SORT Baustein. Dies zeigt, daß die Leistungsbewertung der funktionalen Beschreibung nicht allein auf der Basis der zur Berechnung einer Funktion notwendigen Schritte vorgenommen werden kann, sondern eine exakte Spezifikation der Zeiten für die einzelnen Zellen des Entwurfs wesentlich ist.

2.3. ENTWURF ELEMENTARER ZELLEN

Ausgehend von den logischen Gleichungen der funktionalen Beschreibung ist es einfach Schaltbilder der elementaren Zellen zu entwerfen. Der kreative Teil der Realisierung liegt in der Umsetzung dieser Schaltbilder der Gatterebene in topologische Strukturen der verwendeten Technologie. Dabei werden die Anordnung von Transistoren und Leitungen in der Ebene vorgenommen. Eventuelle Kreuzungen von Leiterbahnen müssen durch einen Wechsel der Ebenen mit entsprechenden Kontakten gelöst werden. Die topologische Darstellung als Strichdiagramm ('stick diagram' siehe MAVOR 83) enthält keine Angaben über die Dimensionierung von Transistoren und die Breite von Leiterbahnen. Diese werden aufgrund der elektrischen Entwurfsregeln bestimmt. Unter Einbeziehung dieser Regeln wird dann aus der Skizze der Topologie das Layout der Zelle entwickelt. Die dabei verwendeten Transistor- und Leitungsdimensionen werden als Parameter bei der elektrischen Simulation verwendet (Abschnitt 2.4). Zeigen die Simulationsergebnisse dann noch ein unbefriedigendes Signalverhalten, muß eine entsprechende Korrektur des Layouts vorgenommen werden.

2.3.1. Register und Vergleicher

Die Verwendung quasistatischer Register mit einem Takt zum Einspeichern neuer und einem zur Wiederauffrischung bereits gespeicherter Informationen ermöglicht eine einfache Synchronisierung des Entwurfs durch eine Zweiphasenuhr (MEAD 80). Das in Bild 2.4 dargestellte Schaltbild und das dazugehörige Strichdiagramm zeigen die beiden Eingangsleitungen des Registers. Da beim Vertauschen der Registerinhalte entweder der obere oder der untere Registerinhalt geladen wird ist es notwendig zwei Datenpfade vorzusehen. Da diese nur alternativ freigegeben werden, sind die Signale UPEN und DOWNEN als Passtransistoren zum Sperren des Datenpfades realisiert. Diese einfache Möglichkeit einen Transistor als Schalter zu verwenden, kann in NMOS-Technik besonders platzsparend realisiert werden.

Da für die Vergleichsfunktion der Registerinhalt negiert und nicht-negiert zur Verfügung stehen muß, ist der Anteil der Ausgangsleitungen im Register relativ hoch. Die Ein- und Ausgangsanschlüsse des Registers

sind so angeordnet, daß die Vergleichierzelle durch direktes Anfügen verdrahtet werden kann.

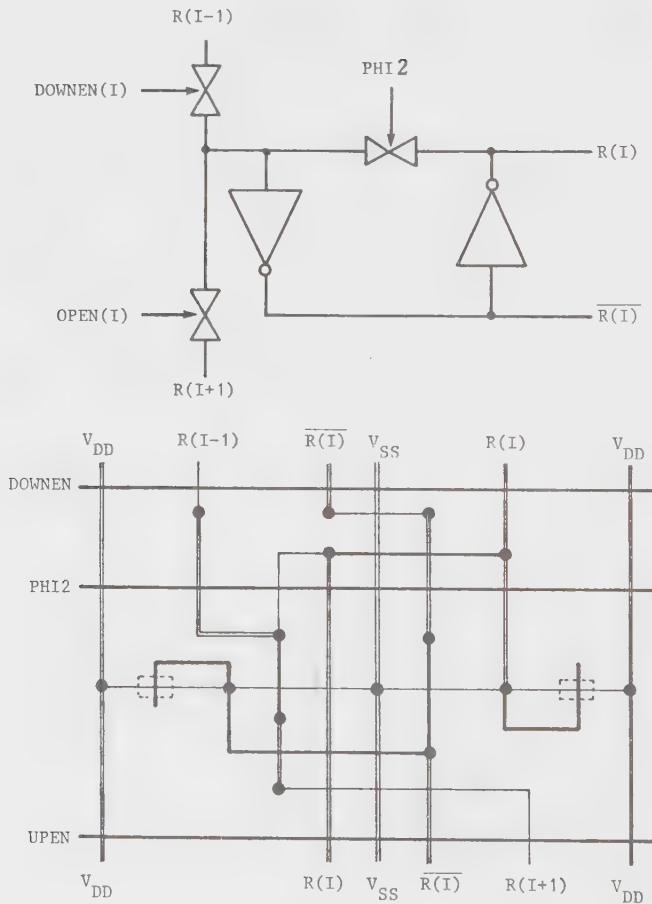


Bild 2.4: Schaltbild und Strichdiagramm einer Registerzelle

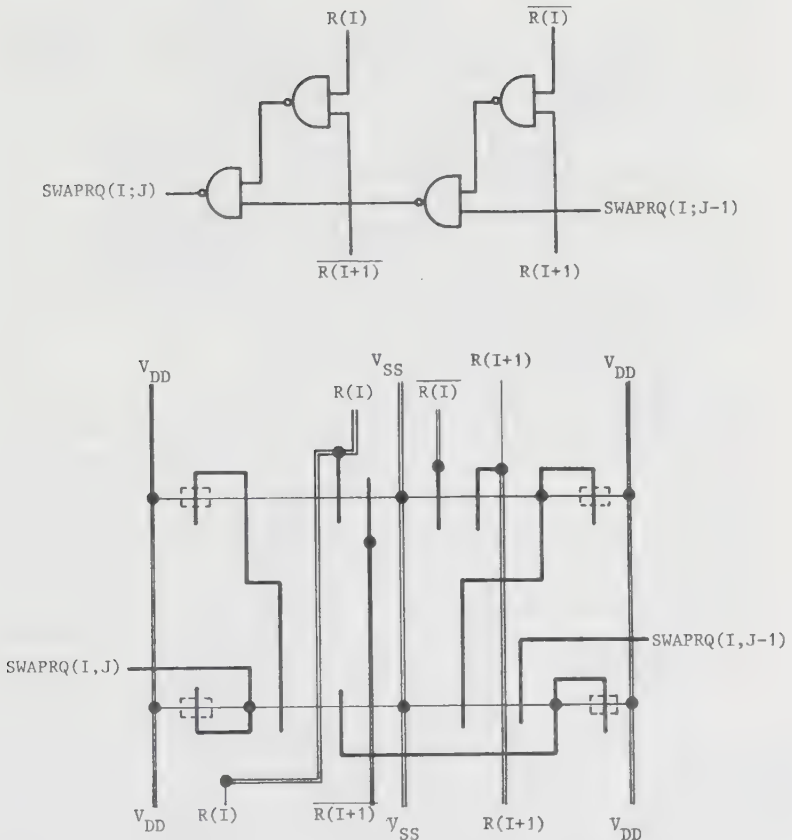


Bild 2.5: Schaltbild und Strichdiagramm einer Vergleichierzelle

Die Vergleichierzelle ist ein Beispiel für die direkte Umsetzung der funktionalen Beschreibung in ein Schaltbild der Gatterebene. Die symmetrische Anordnung der vier NAND Gatter ergibt eine topologische Übereinstimmung der Spannungsversorgung mit der bei den Registern. Prinzipiell werden Signalleitungen und Spannungsversorgung orthogonal

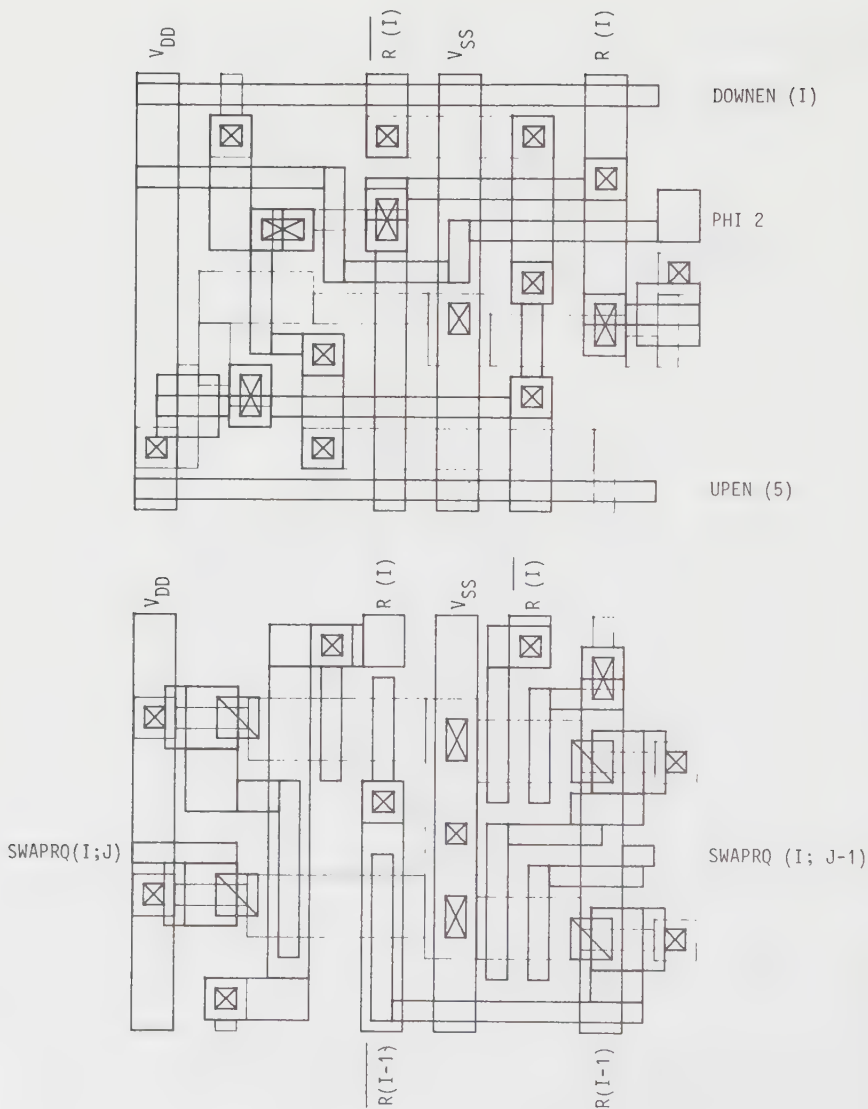


Bild 2.6: Layout von Register- und Vergleichierzelle

angeordnet und in verschiedenen Maskenebenen realisiert (Polysilizium und Aluminium). Beginnt man beim topologischen Entwurf mit der Anordnung der Leitungen und platziert dann die Transistoren, erhält man im allgemeinen ein wesentlich kompakteres Layout (ANCEAU 82). Dies wird durch die Verwendung vergrabener Kontakte (buried contact) unterstützt, die wie Transistoren direkt unter Aluminiumleiter gelegt werden können, ohne von diesen beeinflußt zu werden.

Da in den Vergleichierzellen die Werte beider Register verwendet werden, ist es naheliegend die Datenleitungen so zu verlängern, daß sie bis zum Eingang des nächsten Registers führen (Bild 2.5). Auf diese Weise sind keine gesonderten Datenpfade für den Austausch der Registerinhalte notwendig, vielmehr sind die Vergleichierzellen für den Datenpfad transparent.

Das Layout von Registern und dazwischenliegenden Vergleichierzellen gestattet nicht nur ein Aneinanderfügen in vertikaler, sondern auch in horizontaler Richtung (Bild 2.6). Dazu wurden die Signalleitungen so angeordnet, daß sie am linken und rechten Rand jeweils die gleiche Höhe aufweisen. Da beide Zellen außerdem links und rechts eine vertikale Spannungsleitung haben, kann diese von jeweils zwei nebeneinanderliegenden Zellen gemeinsam genutzt werden. Dazu werden die Zellen überlappend angeordnet und am äußersten rechten Rand mit einer vertikalen Aluminiumbahn ergänzt.

2.3.2. Lokale Steuerlogik

Die lokale Steuerlogik hat die Aufgabe, den Datentransfer zwischen den Datenregistern in allen Bearbeitungsphasen zu steuern. Die Signale für das Kellern und Entkellern (PUSH und POP) werden in der globalen Logik erzeugt und dienen der lokalen Logik als Steuersignale. Die in Bild 2.7 dargestellte Verknüpfung für die Steuerung des Datentransfers von unten nach oben erzeugt das Signal UPIN wenn entkellert wird (POPN=0) oder während der Sortierphase eine Reihenfolgeverletzung vorliegt (SWAP=1). Da die Vergleichierlogik auch während der Ladephase (PUSHN=0) arbeitet, muß das UPIN-Signal dann mit einem Passtransistor unterbrochen werden.

Da das Steuersignal für den Datentransfer über einen längeren Zeitraum an den Registern anliegen muß, wird es in einem Steuerregister gespeichert. Das in Bild 2.8 dargestellte Steuerregister arbeitet zu den

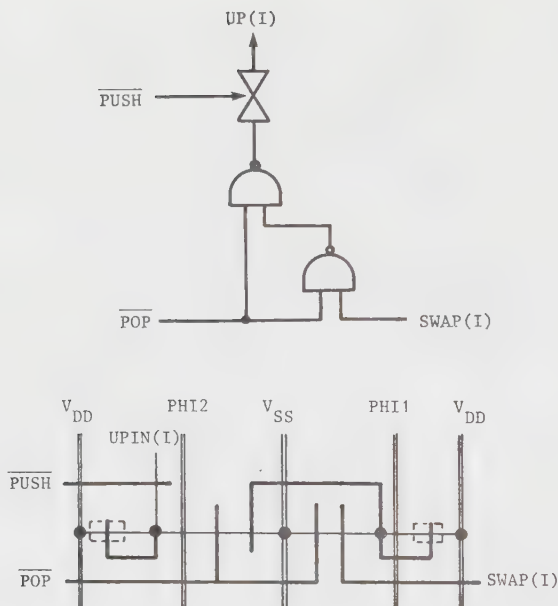


Bild 2.7: Schaltbild und Strichdiagramm lokale Steuerlogik

Datenregistern um einen Takt verschoben, so daß eine Entkopplung von Datentransfer und Vergleich mit der Steuersignal Generierung gewährleistet ist. Die Übernahme neuer Steuerinformation ($\Phi I2$) ist von der Ansteuerung der Datenpfade ($UPEN$) durch einen Passtransistor ($\Phi I1$) entkoppelt.

Die Steuerleitungen für die Eingänge der Datenregister ($UPEN$, $DOWNEN$) treiben Passtransistoren an allen Zellen eines Registers. Dazu sind lange Leitungen (Polysilizium) notwendig die einen hohen Widerstand und parasitäre Kapazitäten aufweisen. Um eine hohe Schaltgeschwindigkeit und hinreichend steile Signalfanken zu erreichen, werden deshalb Pufferverstärker zwischengeschaltet. Schaltbild und Strichdiagramm sind in Bild 2.9 dargestellt. Die Verstärkung von Steuersignalen mit Pufferverstärkern ist charakteristisch für Steuersignale, die aus logischen Verknüp-

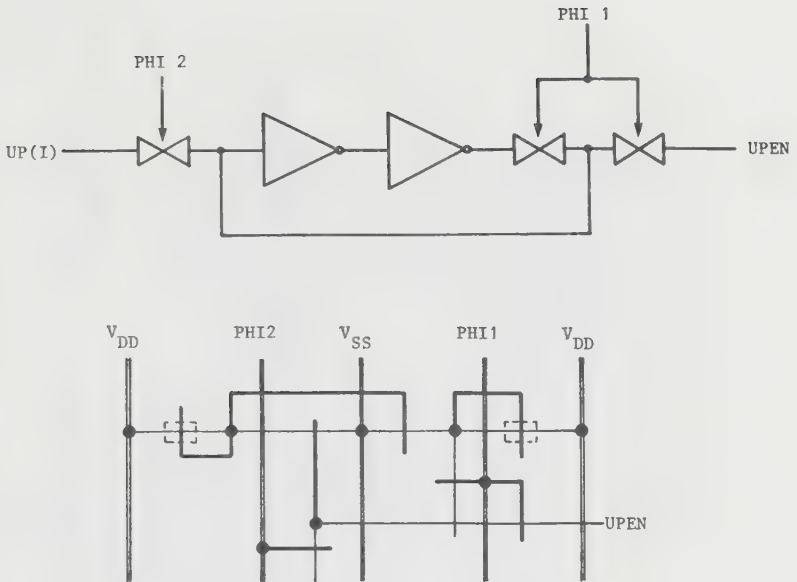


Bild 2.8: Schaltbild und Strichdiagramm Steuerregister

fungen abgeleitet werden.

Die Steuerlogik der Datenpfade ist für beide Richtungen des Datentransfers völlig gleich, wenn die Eingangssignale POP und PUSH vertauscht werden. Das Layout der lokalen Steuerlogik ist deshalb symmetrisch. Die Zellen wurden nur einmal entworfen und sowohl in der ursprünglichen Form, als auch gespiegelt verwendet. Die Logik zur Lösung von Konfliktsfällen beim Sortieren (vergleiche Beispiel 2.1) ist in Bild 2.10 zwischen den Steuerregistern und den Pufferverstärkern deutlich zu erkennen. Auffallend ist auch hier die konsequente Orthogonalität von Daten und Steuerleitungen.

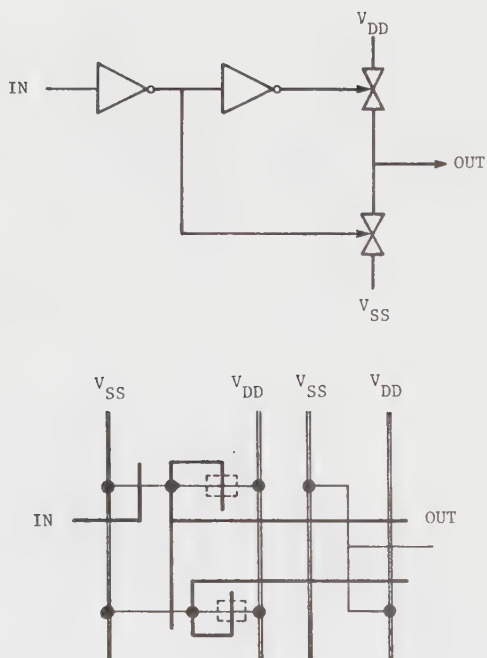


Bild 2.9: Pufferverstärker für Steuersignale

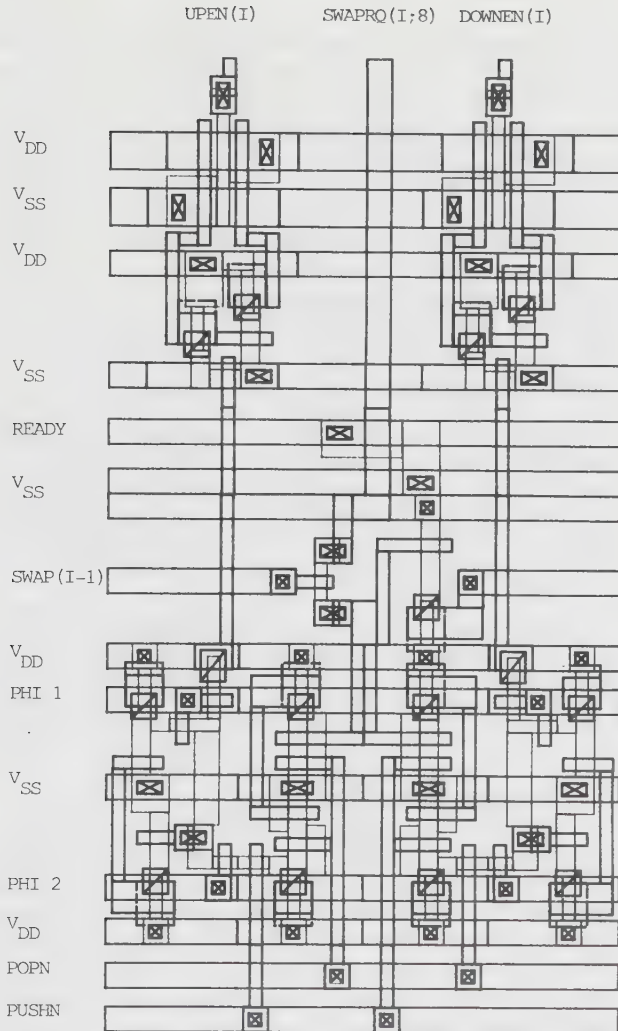


Bild 2.10: Layout der lokalen Steuerlogik

2.3.3. Globale Steuerlogik

Die globale Steuerlogik faßt alle externen Steuer- und Taktsignale, die von den Eingängen des Bausteines kommen, zusammen und verstärkt diese über Pufferverstärker. Die Steuerung der verschiedenen Arbeitsphasen erfolgt über CHIPSEL und READ. Wird der Baustein von der Systemumgebung nicht angesprochen (CHIPSEL=0), wird der Inhalt des Registerfeldes automatisch sortiert. Wenn alle Zahlen in aufsteigender Folge vorliegen, geht der Baustein in einen Wartezustand über, in dem lediglich die Registerinhalte regeneriert werden. Die Ladephase kann beliebig unterbrochen werden, wodurch es möglich ist, das Kellern von Daten mit einer geringeren Datenrate durchzuführen als dies die Zykluszeit des Bausteines gestatten würde. Dies ist bei der Verarbeitung von Daten, die von einem Externspeicher kommen, wichtig. Wurden die gekellerten Daten sortiert (READY=1), kann das Entkellern vorgenommen werden.

Die Generierung der Steuersignale erfolgt über NAND Gatter mit nachgeschalteten Pufferverstärkern. Wegen der einfachen Struktur wurde auf die Angabe von Schaltbild und Strichdiagramm verzichtet und in Bild 2.11 das Layout angegeben. Bemerkenswert ist die Verwendung vergrabener Kontakte für den Übergang zwischen Diffusion und Polysilizium, die teilweise unter den Signalleitungen angeordnet sind.

Für den Test der Schaltung ist die vollständige Kontrolle der Arbeitsphasen durch externe Signale von ausschlaggebender Bedeutung. So können die Register durch Kellern und Entkellern ohne Sortierphase (CHIPSEL ständig 1) getestet werden, ohne daß fehlerhafte Vergleiche diesen Testzyklus beeinträchtigen.

2.3.4. Layout des SORT Bausteines

Das Layout des gesamten SORT Bausteins erhält man durch reguläre Anordnung der einzelnen Register, Vergleiche und lokalen Steuereinheiten, wie dies in der funktionalen Spezifikation durch die Schachtelung der Replikationsoperatoren ('forall...is...endall') festgelegt wurde. An der Peripherie des Chips werden die Anschlußpunkte für Ein- und Ausgabe platziert und mit den Eingängen der entsprechenden Zellen verdrahtet. Diese Peripheriezellen entstammen der Zellbibliothek des CMP'83 (GUYOT 83) und enthalten an den Eingängen bereits Schutztransistoren für Überspannungen und Treiberstufen für die Ausgänge.

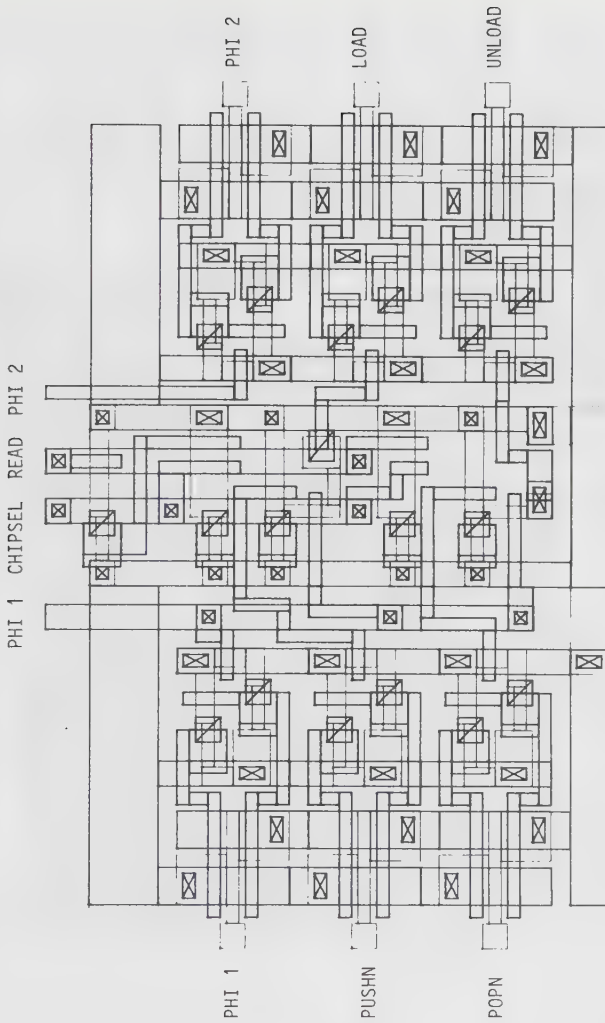


Bild 2.11: Layout der globalen Steuerlogik

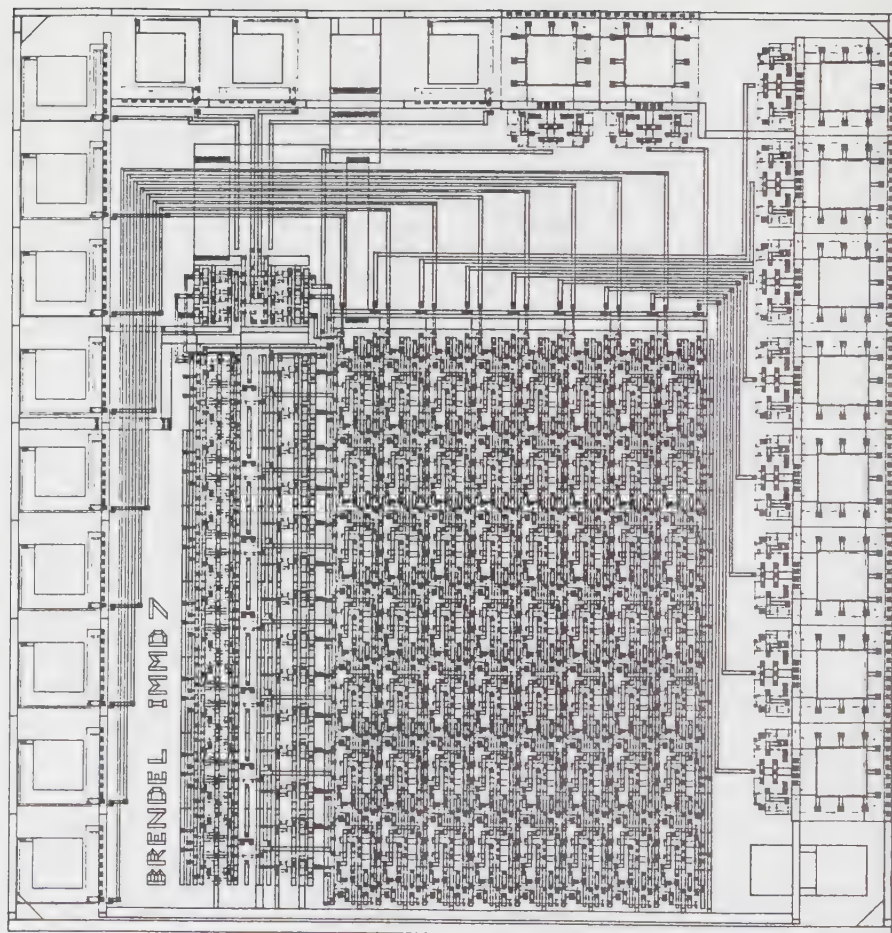


Bild 2.12: Layout des SORT Bausteins

2.4. ELEKTRISCHE SIMULATION

Die Berechnung des elektrischen und zeitlichen Verhaltens eines Entwurfs durch Simulation auf der Technologieebene ist nur für einfache Zellen möglich. Der enorme Rechenaufwand gestattet lediglich Simulationen bis zu einer Anzahl von etwa 100 Transistoren und wenigen Millisekunden Signaldauer. Wurde, wie beim BUBBLE SORT Baustein, eine funktionale Simulation durchgeführt, kann sich die elektrische Simulation auf die Validierung des Signalverhaltens und der angenommenen Verzögerungszeiten beschränken.

Da das zeitliche Verhalten der einzelnen Zellen von den Transistordimensionen, den Leitungswiderständen und parasitären Kapazitäten bestimmt wird, ist eine Beschreibung auf der Transistorebene erst nach Erstellung des Layouts möglich. Die ermittelten Verzögerungszeiten für einzelne Zellen des Entwurfs werden mit den, in der funktionalen Spezifikation angenommenen, verglichen. Werden diese Zeitbedingungen nicht eingehalten, ist entweder eine Verbesserung der Schaltung durch äquivalente Umformungen oder eine Optimierung des Layouts zur Reduktion von Kapazitäten notwendig. Ergibt eine erneute Simulation immer noch keine ausreichenden Schaltgeschwindigkeiten, müssen Taktraster und Zeitdeklarationen in der funktionalen Beschreibung überarbeitet werden. Dies kann einen vollständigen Neuentwurf notwendig machen.

Im folgenden Abschnitt werden die Ergebnisse der elektrischen Simulation für die einzelnen Zellen des BUBBLE SORT Bausteins diskutiert und abschließend mit der funktionalen Spezifikation verglichen.

2.4.1 Physikalische Parameter für SPICE

Die physikalischen Parameter des im CMP'83 verwendeten NMOS Prozesses (ANCEAU 81) sind die Grundlage für die Ermittlung der Modellparameter für die elektrische Simulation. Der Flächenwiderstand und die Kapazität der einzelnen Materialien wird zur Bestimmung der parasitären Kapazitäten der Zellelemente verwendet.

Für die elektrische Simulation wurde das weit verbreitete Simulations-

paket SPICE der 'University of California at Berkeley' eingesetzt (NEWTON 81). Eine Beschreibung der in SPICE implementierten MOS-Transistormodelle enthält (ALEXANDER 77), dort sind auch die wichtigsten Modellparameter und ihre gegenseitige Abhängigkeit beschrieben. Die Bestimmung der Modellparameter aus den häufig nur unvollständig bekannten physikalischen Prozessparametern ist äußerst schwierig, da die Konvergenz der bei der Simulation eingesetzten Iterationsverfahren nur für bestimmte Parameterbereiche gewährleistet ist. Die Zuverlässigkeit der Simulationsergebnisse hängt wesentlich von der Wahl der Parameter ab, diese sind jedoch nicht für einen größeren Bereich der Transistordimensionen konstant. Eine Parameteroptimierung ist nur durch die Fertigung von Testschaltungen und eine laufende Prozesskontrolle mit einer ständigen Anpassung der Simulationsparameter möglich (BRANDT 83). Da die im folgenden diskutierten Simulationsergebnisse relativ gut mit den Erfahrungen bei CMP'82 übereinstimmen, sind keine wesentlichen Abweichungen zum realen Schaltkreis zu erwarten.

PARAMETER	VERARMUNGSTYP	ANREICH.TYP
Schwellspannung (V)	VTO: 0.6	-1.4
Substrat Konstante	GAMMA: 0.0	0.2
Sättigungsstrom (A/m^2)	JS: $1.0E-8$	$1.0E-8$
Oberflächenmobilität ($cm^2/V-s$)	UO: 800	800
Flächenwiderstand ($\Omega m/sq$)	RSH: 30.0	30.0
Gate-Source Überlappungskapazität (F/m)	CGSO: $4.5E-10$	$4.5E-10$
Gate-Drain Überlappungskapazität (F/m)	CGDO: $4.5E-10$	$4.5E-10$
Seitenwandkapazität (F/m)	CJSW: $1.0E-9$	$1.0E-9$

Tabelle 2.1: SPICE Modellparameter für CMP'83

Zur Berechnung der parasitären Leitungskapazitäten ist es sinnvoll, eine Umrechnung der physikalischen Kapazitätsparameter in Standardflächeneinheiten vorzunehmen. Da das Layout in LAMBDA Einheiten entworfen wird ($\text{LAMBDA} = 3 \mu\text{m}$), ist dadurch eine einfachere Kapazitätsberechnung möglich. Die Leitungskapazitäten und Widerstände werden als zusätzliche Elemente in die Simulationsschaltung eingebaut, um eine genaue Simulation des Signalverhaltens zu ermöglichen.

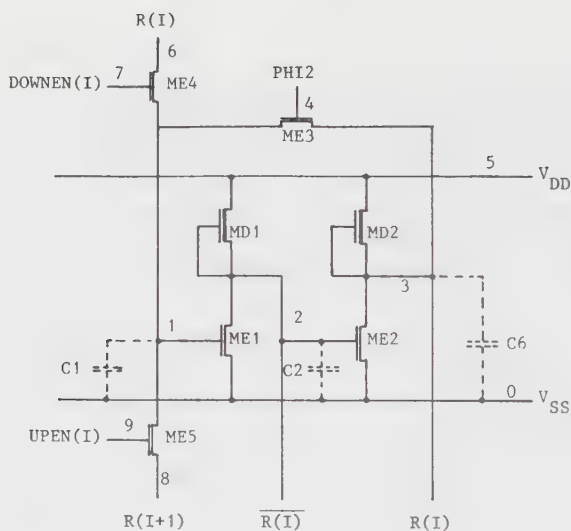
Leitungsmaterial	Flächenkapazität in $9 \mu\text{m}^2$
Gate - Transistorkanal	$36.0 \text{ E-4 pF}/\lambda^2$
Diffusionsgebiete	$4.5 \text{ E-4 pF}/\lambda^2$
Diffusionsrand	$15.0 \text{ E-4 pF}/\lambda$
Polysilizium	$3.6 \text{ E-4 pF}/\lambda^2$
Aluminium	$2.7 \text{ E-4 pF}/\lambda^2$

Tabelle 2.2: Flächenkapazitäten

2.4.2. Register und Vergleicher

Aus dem in Bild 2.4 angegebenen Strichdiagramm für die Registerschaltung kann das in Bild 2.13 dargestellte Transistorschaltbild gewonnen werden. Die Beschreibung für die elektrische Simulation besteht aus den Transistornamen, den Anschlußknoten, dem Transistortyp, den Transistordimensionen und den Flächen der Diffusionsgebiete. Das Transistornetz ergibt sich aus den durch die Knotennummern definierten gemeinsamen Anschlußpunkten.

Da das zeitliche Verhalten des Registers durch die Kapazitäten der Leitungen stark beeinflusst wird, ist es notwendig, zusätzliche Parasitärkapazitäten C1, C2 und C3 einzufügen. Die Kapazitäten wurden aus den



```

.SUBCKT REG 2 3 4 5 6 7 8 9 1
*parameter: -Q, Q,refresh,vdd, ind,down, inup, up,TEST
MD1 5 2 2 0 DMOD L=6U W=6U AD=240P AS=225P NRD=2 NRS=3
ME1 2 1 0 0 EMOD L=6U W=15U AD=225P AS=700P NRD=2 NRS=6
MD2 5 3 3 0 DMOD L=6U W=6U AD=150P AS=150P NRD=2 NRS=3
ME2 3 2 0 0 EMOD L=6U W=15U AD=360P AS=360P NRD=1 NRS=1
ME3 3 4 1 0 EMOD L=6U W=6U
ME4 6 7 1 0 EMOD L=6U W=6U
ME5 8 9 1 0 EMOD L=6U W=6U
CPAR1 1 0 0.17P
CPAR2 2 0 0.27P
CPAR3 3 0 0.18P
.ENDS REG

```

Bild 2.13: Transistorschaltung und SPICE-Modell Register

im Layout vorhandenen Flächen mit Hilfe der Tabelle 2.2 berechnet (C1=0.17pF, C2=0.27pF, C3=0.18pF).

Die elektrische Simulation wird durch Eingangsspannungen an den Passtransistoren für Registereingang und Wiederauffrischung angeregt. Das an den Ausgangsleitungen zu beobachtende Signalverhalten (Bild 2.14) zeigt den Einschwingvorgang und die Verzögerungszeit für das Setzen und

Rücksetzen des Registers. Die beiden Takte Φ_{11} und Φ_{12} steuern das Register. Aus den Kurvenverläufen ist deutlich das typische Verhalten von NMOS Invertern zu entnehmen, bei denen der Einschaltvorgang länger dauert als der Ausschaltvorgang ($t_1=60\text{ns}$, $t_2=40\text{ns}$).

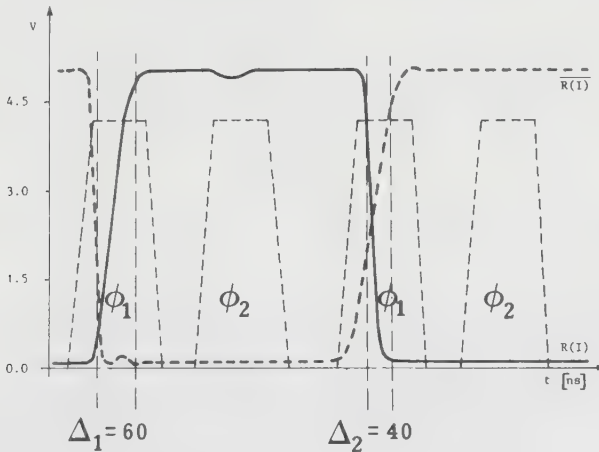
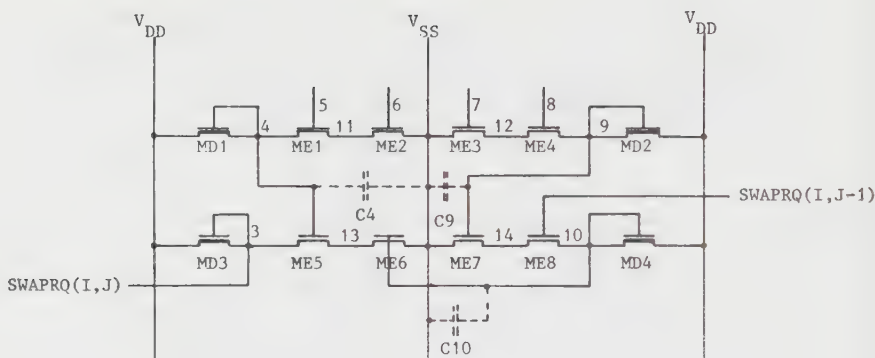


Bild 2.14: Signalverhalten Register

Ähnlich wie beim Register wird das aus dem Strichdiagramm gewonnene Transistorschaltbild der Vergleichszellen um parasitäre Kapazitäten ergänzt (Bild 2.15). Die Kanalbreite der Verarmungstransistoren ist wegen der bei einem NAND-Gatter in Serie geschalteten Transistoren doppelt so groß wie beim Datenregister. Dadurch wird etwa das zeitliche Verhalten eines Inverters erreicht.

Beim Signalverhalten der Vergleicherschaltung (Bild 2.16) muß zwischen zwei verschiedenen Verzögerungszeiten unterschieden werden. Die Zeit zur Bildung des Vergleichs zwischen den beiden Registerbits ($t_v=10\text{ns}$) bestimmt die Verzögerungszeit zur Erzeugung der Tauschanforderung. Dieser Signalübergang tritt genau dann auf, wenn das Register $R(I)$ auf der aktuellen Bitposition größer ist als $R(I+1)$. Das gesamte Zeitverhalten der Vergleicherschaltung wird jedoch durch die Signallaufzeit des Übertrags aus niederwertigen Bitpositionen bestimmt ($t_s = 20\text{ ns}$). Im ungünstigsten Fall muß der Übertrag von der niederwertigsten Bitposition



```

.SUBCKT COM 1 2 3 5 6 7 8
*parameter vdd,SWAPRQ(I-1),SWAPRQ(I),RC(I),RC(I+1),RC(I),RC(I+1)
MD1 1 4 4 0 DMOD L=9U W=6U AD=160P AS=235P NRD=1 NRS=1
ME1 4 5 11 0 EMOD L=6U W=18U AD=160P AS=80P NRD=1 NRS=1
ME2 11 6 0 0 EMOD L=6U W=18U AD=80P AS=610P NRD=1 NRS=1
ME3 12 7 0 0 EMOD L=6U W=18U AD=64P AS=290P NRD=1 NRS=1
ME4 9 8 12 0 EMOD L=6U W=18U AD=108P AS=64P NRD=1 NRS=1
MD2 1 9 9 0 DMOD L=9U W=6U AD=160P AS=72P NRD=1 NRS=1
MD3 1 3 3 0 DMOD L=9U W=6U AD=160P AS=108P NRD=1 NRS=1
ME5 3 4 13 0 EMOD L=6U W=18U AD=108P AS=190P NRD=1 NRS=1
ME6 13 10 0 0 EMOD L=6U W=18U AD=190P AS=490P NRD=1 NRS=1
ME7 14 9 0 0 EMOD L=6U W=18U AD=54P AS=270P NRD=1 NRS=1
ME8 10 2 14 0 EMOD L=6U W=18U AD=162P AS=54P NRD=1 NRS=1
MD4 1 10 10 0 DMOD L=9U W=6U AD=162P AS=54P NRD=1 NRS=1
C4 4 0 0.065P
C9 9 0 0.064P
C10 10 0 0.073P
.ENDS COM

```

Bild 2.15: Transistorschaltung und SPICE-Modell Vergleich

durch die folgenden Vergleichszellen erfolgen. Dies ergibt eine Gesamtzeit für einen Vergleich mit acht Bit von 160ns.

Der Signalverlauf für eine Vergleicherschaltung mit vier Bit (Bild 2.17) zeigt die Fortpflanzung der Vertauschungsanforderung (SWAPRQ) durch

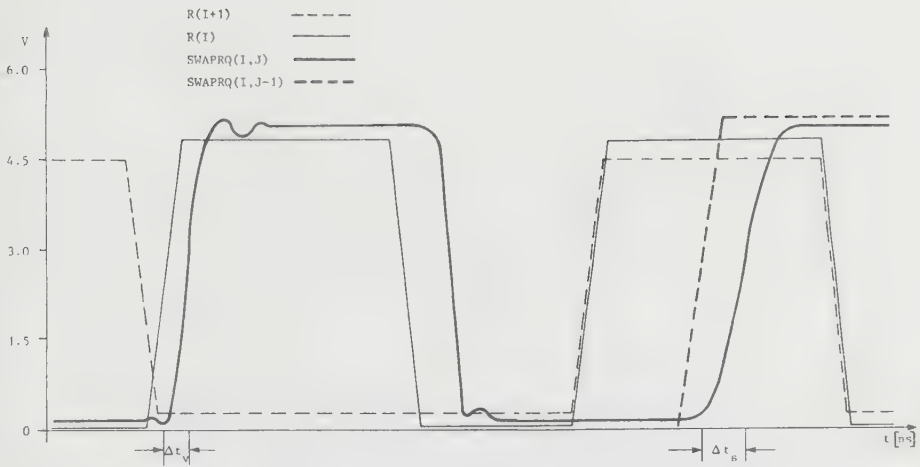


Bild 2.16: Signalverhalten Vergleicher

mehrere Stufen des Vergleichers. Im ersten Fall ($SWAPRQ(I;1) = 1$) ist das Register $R(I+1)$ größer als $R(I)$, so daß die Vertauschungsanforderung der ersten Bitposition durch die weiteren Stufen unterdrückt wird. Im zweiten Fall ist $R(I)$ größer als $R(I+1)$, weshalb in der zweiten Bitposition ($SWAPRQ(I;2) = 1$) eine Vertauschungsanforderung erzeugt wird, die sich durch die weiteren Stufen fortpflanzt. Da der Übergang vom Null- in den Eins-Zustand länger dauert als umgekehrt, sind die Signalverläufe beim Erzeugen der Anforderung flacher als umgekehrt. Aufgrund der niedrigen Schwellspannungen der Transistoren sind die Phasen der einzelnen Vergleicherstufen jedoch überlagert, so daß die Gesamtzeit der Verzögerung des Übertrags nicht wesentlich verlängert wird.

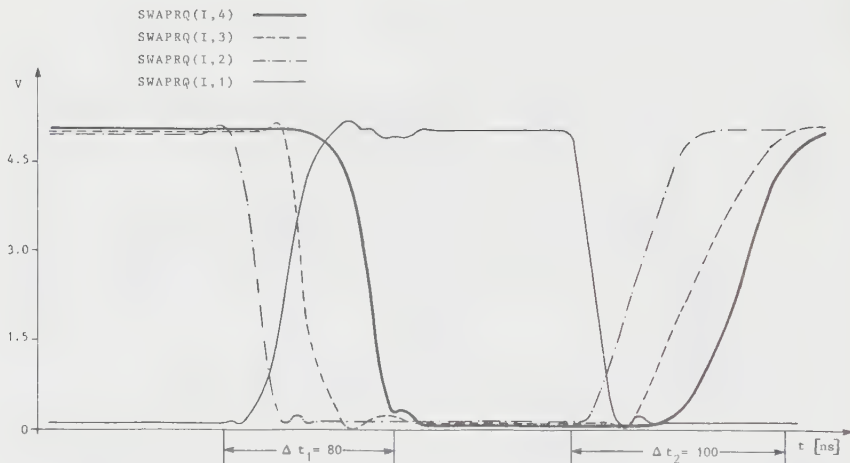
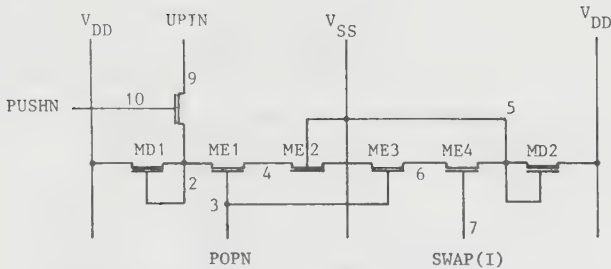


Bild 2.17: Fortpflanzung des Vergleicherausgangs

2.4.3. Lokale und globale Steuerlogik

Die Steuerung des Datentransfers zwischen den Registern erfolgt durch die lokale Steuerlogik. Da die beiden Transferrichtungen und die steuernde Steuerlogik bis auf die Vertauschung der Steuersignale POPN und PUSHN gleich sind, genügt es, die Zellen für die Steuerung des Datentransfers von Registern mit höherem Index zu Registern mit niedrigerem Index zu betrachten. Die im Datenpfad zwischen den Registern zwischengeschalteten Passtransistoren (siehe Bild 2.1) werden vom Bearbeitungszustand geschaltet. So wird beim Entkellern der sortierten Daten in der lokalen Steuerlogik das Steuersignal UPEN erzeugt, wenn in der globalen Steuerlogik das Signal zum Entkellern POPN generiert wird. Während der Sortierphase (POPN=PUSHN=1) wird der Datenpfad zwischen zwei Registern genau dann freigegeben, wenn die Vergleiche eine Verletzung der Reihenfolgebedingung feststellen (SWAPRQ=1). Da der Vergleich auch beim Kellern stattfindet, muß das Steuersignal UPEN dann unterdrückt werden. Das aus dem Layout der lokalen Steuerlogik (Bild 2.10) gewonnene Transistorschaltbild ist mit der SPICE-Beschreibung in Bild 2.18 dar-

gestellt.



```

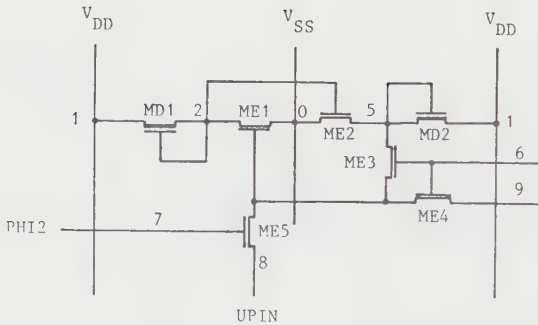
.SUBCKT UPCTRL 1 3 7 9 10
*PARAMETER VDD, POPN, SWAP(I), UPIN, PUSHN
MD1 1 2 2 0 DMOD L=9U W=6U AD=162P AS=90P
ME1 2 3 4 0 EMOD L=6U W=18U AD=220P AS=100P
ME2 4 5 0 0 EMOD L=6U W=18U AD=100P AS=220P
MD2 1 5 5 0 DMOD L=9U W=6U AD=162P AS=90P
ME4 5 7 6 0 EMOD L=6U W=18U AD=160P AS=80P
ME3 6 3 0 0 EMOD L=6U W=18U AD=80P AS=220P
ME5 2 10 9 0 EMOD L=6U W=6U AD=100P AS=35P
C9 9 0 0.01P
.ENDS UPCTRL

```

Bild 2.18: Transistorschaltung der lokalen Steuerlogik

Die Steuersignale UPEN und DOWNEN werden während der Taktphase PHI2 erzeugt und erst zur nächsten Taktphase PHI1 zur Steuerung der Eingänge der Datenregister benötigt. Um stabile Steuersignale zu gewährleisten, müssen diese deshalb in Registern zwischengespeichert werden. Die Steuerregister haben prinzipiell die gleiche Schaltung wie die Datenregister jedoch ein einfacheres Layout, da kein Datenpfad integriert ist. Die in Bild 2.19 dargestellte Transistorschaltung enthält deshalb auch keine parasitären Kapazitäten.

Da die Steuersignale für die Datenpfade lange Leitungen in Polysilizium und für jedes Bit der Datenregister einen Passtransistor treiben müssen, ist es notwendig diese Signale durch Treiberschaltungen zu verstärken. Wesentlich für die Zuverlässigkeit des Schaltkreises ist dabei die Steilheit der Signalflanken, da der Datenaustausch nur während einer relativ kurzen Zeitspanne stattfinden kann. Um eine realistische Simulation zu erreichen, wurde in das in Bild 2.20 dargestellte Transistor-



```

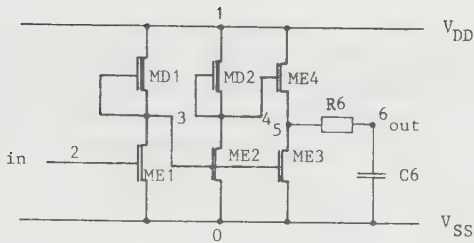
.SUBCKT UPREG 1 6 7 8 9
*PARAMETER VDD,PHI1,PHI2,UPIN,UPEN
MD1 1 2 2 0 DMOD L=9U W=6U AD=162P AS=90P
ME1 2 3 0 0 EMOD L=6U W=9U AD=90P AS=300P
MD2 1 5 5 0 DMOD L=9U W=6U AD=162P AS=90P
ME2 5 2 0 0 EMOD L=6U W=9U AD=190P AS=135P
ME3 5 6 3 0 EMOD L=6U W=6U AD=100P AS=380P
ME4 3 6 9 0 EMOD L=6U W=6U AD=110P AS=120P
ME5 8 7 3 0 EMOD L=6U W=6U AD=20P AS=270P
C3 3 0 0.076P
C9 9 0 0.063P
.ENDS UPREG

```

Bild 2.19: Transistorschaltung Steuerregister

schaltbild eine Ersatzdarstellung der Steuerleitungen aufgenommen. Der Leitungswiderstand R6 wurde mit einer der Leitungslänge und den Dimensionen der Passtransistoren entsprechenden parasitären Kapazität C6 abgeschlossen. Die Platzierung der Kapazität nach dem Widerstand ist eine pessimistische Annahme und läßt im Schaltkreis ein besseres Signalverhalten als in der Simulation erwarten.

Die Generierung von Steuersignalen in der lokalen Logik ist für alle drei Bearbeitungsphasen in Bild 2.21 und 2.22 dargestellt. Dabei wurde von der in der funktionalen Spezifikation festgelegten Zykluszeit von 500 ns ausgegangen. In der Taktphase PHI1 findet prinzipiell der Datenaustausch zwischen den Registern statt, während bis zum Ende der Taktphase PHI2 die Bildung des Vergleichsergebnisses und die Auswertung durch die lokale Steuerlogik erfolgt. Das am Ende dieser Taktphase



```

.SUBCKT SUPER 1 2 6
*PARAMETER VDD, IN,OUT
MD1 1 3 3 0 DMOD L=9U W=6U AD=270P AS=180P
ME1 3 2 0 0 EMOD L=6U W=9U AD=54P AS=135P
MD2 1 4 4 0 DMOD L=9U W=6U AD=370P AS=45P
ME2 4 3 0 0 EMOD L=6U W=9U AD=54P AS=189P
ME4 1 4 5 0 EMOD L=6U W=24U AD=360P AS=162P
ME3 5 3 0 0 EMOD L=6U W=24U AD=162P AS=360P
R6 5 6 7.5E3
C6 6 0 0.3P
.ENDS SUPER

```

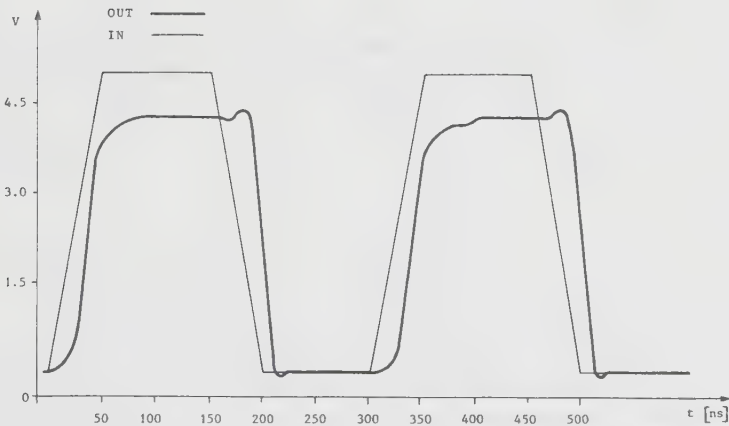


Bild 2.20: Schaltung und Simulation der Leitungstreiber

ermittelte Steuersignal wird dann in die Steuerregister übernommen und dient im nächsten Zyklus zur Steuerung des Datentransfers während der

Taktphase PH11. Die Signale POPN und PUSHN der globalen Steuerlogik müssen während der gesamten zweiten Taktphase anliegen und bestimmen die Operation des Schaltkreises im nächsten Zyklus.

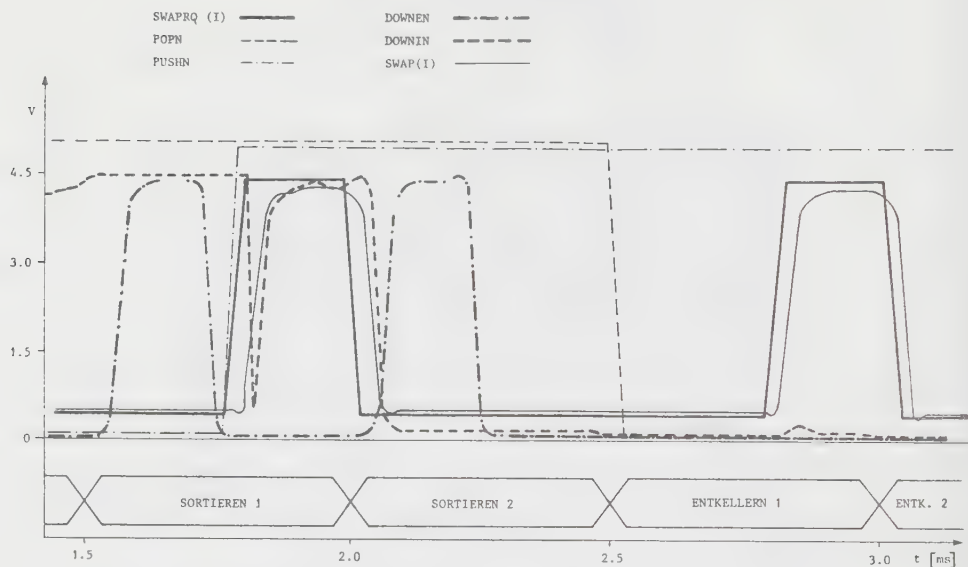


Bild 2.21: Erzeugung des Steuersignals DOWNEN

Beim Kellern von Daten (POPN=1, PUSHN=0) wird in der lokalen Steuerlogik das Signal DOWNEN erzeugt, das den Datentransfer in der ersten Sortierphase bestimmt (siehe Bild 2.21). Die am Ende dieser Sortierphase von der Vergleichierlogik festgestellte Reihenfolgeverletzung (SWAPRQ=1) verursacht das Setzen beider Steuerregister (UPIN=DOWNIN=1). Das Vertauschen der Inhalte der Datenregister findet dann in der zweiten Sortierphase statt (siehe Bild 2.21 und 2.22). Da in der zweiten Sortierphase dann die korrekte Reihenfolge der Daten vorliegt, wird von den Vergleichern keine Vertauschung angefordert (SWAPRQ=0). Die Steuerregister werden zurückgesetzt und die Datenpfade zwischen den Datenregistern bleiben gesperrt (UPEN=DOWNEN=0). Im folgenden Zyklus wird durch den Signalwechsel von POPN das Entkellern der Daten erzwungen. Dabei wird

trotz des anliegenden Signals zum Tausch der Datenregister das Signal DOWNIN unterdrückt (Bild 2.21), während UPIN gesetzt wird (Bild 2.22). Die Signalverläufe zeigen, daß die zwischengeschalteten Treiberstufen gute Steuersignale erzeugen sollten.

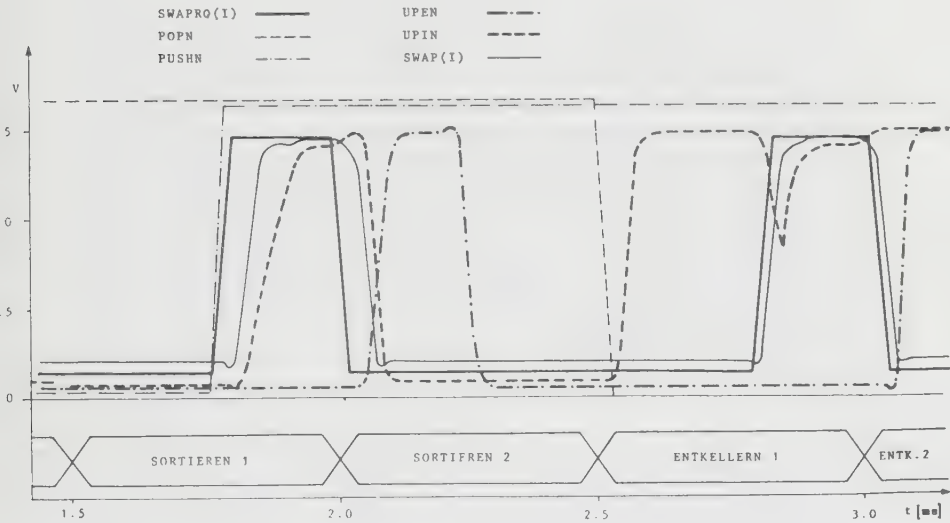


Bild 2.22: Erzeugung des Steuersignals UPIN

2.4.4. Vergleich elektrische und funktionale Simulation

Die detaillierten Ergebnisse der elektrischen Simulation erfordern gegenüber der funktionalen Simulation einen unverhältnismäßig höheren Aufwand. So wird bei der elektrischen Simulation für die Berechnung der in den Bildern 2.21 und 2.22 angegebenen Signalverläufe die gleiche Rechenzeit benötigt wie für die Übersetzung der funktionalen Spezifikation einschließlich der Simulation mehrerer hundert Sortierzyklen (etwa 10 Minuten CPU-Zeit auf einer Rechanlage VAX 11/780). Die elektrische Simulation ist deshalb nur für die Analyse einfacher Zellen

des gesamten Entwurfs sinnvoll einzusetzen. Andererseits sind die in der funktionalen Spezifikation gemachten Annahmen über das zeitliche Verhalten von Registern und Terminals nur durch eine genaue Simulation des elektrischen Verhaltens zu überprüfen, da diese wesentlich von Leitungswiderständen und parasitären Kapazitäten abhängen.

In der funktionalen Spezifikation wurde von einer Gatterschaltzeit von etwa 10 ns ausgegangen, was durch die elektrische Simulation im Durchschnitt bestätigt wurde. Die Berücksichtigung der elektrischen Entwurfsregeln bei der Dimensionierung der Transistoren hat größtenteils zu einer relativ geringen Streuung der Signallaufzeiten geführt. Größere Abweichungen von den ursprünglichen Annahmen bei den Registern machen jedoch die Notwendigkeit einer elektrischen Simulation deutlich. Dabei wirkten sich insbesondere Leitungswiderstände bei langen Signalleitungen und parasitäre Kapazitäten bei komplexen Zellen aus. Die gesamte Zeitbilanz der funktionalen Spezifikation konnte nur deshalb eingehalten werden, da das Taktraster der Spezifikation großzügig ausgelegt wurde und die langsameren Teile der Schaltung nicht auf dem kritischen Pfad der Vergleichsüberträge lagen.

2.5. KRITIK DER FUNKTIONALEN SPEZIFIKATION

Ausgangspunkt beim Entwurf des SORT-Bausteines war die funktionale Spezifikation. Darauf aufbauend wurde der Grundriß des Bausteins, die Schaltung und die topologische Beschreibung der Zellen und zuletzt das Layout entwickelt. Die zuletzt durchgeführte elektrische Simulation diente der Validierung des elektrischen und zeitlichen Verhaltens der einzelnen Zellen. Das globale Zusammenwirken kann aus Aufwandsgründen damit jedoch nicht überprüft werden. Die funktionale Simulation bietet deshalb die einzige Möglichkeit das Zusammenwirken der einzelnen Komponenten zu überprüfen. Die korrekte Modellbildung auf der funktionalen Ebene ist deshalb unabdingbare Voraussetzung für zuverlässige Simulationsergebnisse. Da die funktionale Spezifikation mit Hilfe der Rechner-Entwurfssprache ERES formuliert wurde, liegen der funktionalen Simulation auch die Modellvorstellungen von ERES zugrunde. Elemente hochintegrierter Schaltkreise, die keine direkte Entsprechung in ERES haben, wie dynamische Register und Transferrgatter, wurden durch semantisch ähnliche Elemente in ERES ersetzt. Eine genaue Analyse des integrierten Schaltkreises zeigt aber, daß an diesen Stellen Schwierigkeiten auftreten.

Die dynamischen Register werden durch den Pegel eines Steuersignales zur Übernahme neuer Daten aktiviert. Während dieses Steuersignal anliegt, werden die neuen Eingangsdaten mit einer gewissen Verzögerung an den Ausgang des Registers und von dort in die übrige Schaltung weitergegeben. Erst beim Übergang des Steuersignals vom Eins- in den Nullzustand wird der Registereingang gesperrt und der zuletzt am Eingang anliegende Wert gespeichert. Für eine korrekte Funktion der Register ist eine ausreichend lange Dauer der Taktsignale wichtig. Da jedoch die Eingangsdaten eines Registers stets Ausgangsdaten eines benachbarten Registers sind, besteht bei längeren Taktphasen die Gefahr, daß Änderungen an den Eingängen des Registers kurzzeitig Änderungen der Ausgänge bewirken. Dies führt dazu, daß am Eingang des nächsten Registers kein stabiler Wert mehr anliegt und sich deshalb Signale in einer Taktphase durch mehrere Register fortpflanzen.

Die Modellvorstellung der funktionalen Beschreibung in ERES, daß ein Register zu einem Zeitpunkt sowohl gelesen als auch beschrieben werden

kann, trifft wegen der endlichen Ausdehnung der Taktsignale für die bei integrierten Schaltkreisen üblichen dynamischen Register nicht zu (vergleiche Abschnitt 4.2).

Eine korrekte Funktion des SORT-Bausteins ist deshalb nur bei relativ kurzen Taktsignalen zu erwarten, die wegen der langen Signalleitungen und der zwischengeschalteten Treiberstufen nur schwierig einzuhalten sind. Da sich die Schaltgeschwindigkeit bei Temperaturschwankungen stark ändert, ist ein gewähltes Taktschema nur für einen schmalen Temperaturbereich gültig. Eine wesentliche Erweiterung von ERES ist deshalb die Möglichkeit zur Modellierung verschiedener Registerimplementierungen und ein entsprechendes Taktmodell (vergleiche Abschnitt 5.1). Die in ERES realisierte Modellvorstellung von Takten, die eine impulsförmige Charakteristik aufweisen, ist nur bei flankengesteuerten Registern realistisch.

Eine ähnliche Problematik beinhalten die in hochintegrierten Schaltkreisen häufig verwendeten Transfergatter, die einen Datenpfad freigeben oder sperren. Im Gegensatz zu logischen Verknüpfungen bei denen neue Signale aus Eingangssignalen erzeugt werden, haben diese bidirektionalen Charakter. Wird ein Datenpfad durch ein Transfergatter gesperrt, geht dieser in einen hochohmigen Zustand über und speichert den letzten Wert für eine gewisse Zeit. Dieser hochohmige Zustand und das speichernde Verhalten von Verbindungsleitungen haben in ERES kein entsprechendes Gegenstück. Eine Erweiterung von ERES zur Entwurfssprache für hochintegrierte Schaltkreise muß deshalb entsprechende Modelle enthalten (vergleiche Abschnitt 4.1).

Die elektrische Simulation hat gezeigt, daß eine Zeitbeschreibung auf der funktionalen Ebene wie in ERES nur für größere Teilfunktionen und nicht für einzelne Operationen sinnvoll ist. Die ERES Methode, stets einem logischen Ausdruck (terminal) eine Gesamtzeit zuzuweisen, erleichtert die Validierung durch die funktionale Simulation. Eine scheinbar genauere Spezifikation der Zeiten für einzelne Gatter ist wegen der zum Zeitpunkt der Spezifikation unbekannten Widerstände und Kapazitäten nicht aussagekräftiger. Das asymmetrische Signalverhalten der NMOS-Technologie würde bei einer detaillierten Zeitspezifikation zu pessimistischen Ergebnissen in der funktionalen Simulation führen, da gegenläufige Signalwechsel in komplexeren Verknüpfungen zu insgesamt kürzeren Verzögerungszeiten führen als dies die Summe der Einzelzeiten erwarten läßt.

(siehe Bild 2.17).

Zu den, bei der Realisierung des BUBBLE SORT Bausteins zur Verstärkung der Steuersignale verwendeten Treibern, gibt es in der funktionalen Spezifikation keine entsprechende Deklaration. Für eine automatische Synthese hochintegrierter Schaltungen ist es jedoch unumgänglich, daß jedem technologischen Element ein entsprechendes Objekt in der funktionalen Spezifikation zugeordnet ist. Daten- und Signalleitungen hochintegrierter Schaltkreise haben je nach ihrer Verwendung verschiedene Eigenschaften, die in der Modellbildung auf der Architekturebene berücksichtigt werden müssen. Die in den Abschnitten 4 und 5 behandelten Erweiterungen von ERES sind deshalb Sprachelemente, die zur Beschreibung hochintegrierter Schaltungen notwendig sind.

*The total Silicon Compiler
is not around the corner -
it requires a conceptual
breakthrough to make it a reality.*

B. Murphy

3. ATTRIBUTIERTE GRAPH-GRAMMATIKEN

Die große Anzahl oft informaler Beschreibungsmethoden für die Spezifikation von Eigenschaften und Struktur hochintegrierter Schaltkreise erschweren den Vergleich und den Austausch von Informationen über das funktionale, elektrische und zeitliche Verhalten. Für die Beschreibung und Implementierung eines CAD-Systems zur automatischen Synthese hochintegrierter Schaltkreise ist es jedoch unumgänglich, daß von den theoretischen Modellen der Register-Transferebene bis zum generierten Layout ein konsistentes, formales Beschreibungsmittel verwendet wird.

In diesem Abschnitt werden die wichtigsten Grundlagen aus der Theorie der Graph-Grammatiken und die im weiteren verwendete Notation eingeführt. Seit den ersten Ansätzen (SCHNEIDER 70, NAGL 74) haben die Graph-Grammatiken bereits ein weites Feld von Anwendungen gefunden (NAGL 79). Die Anwendung auf die formale Beschreibung von Programmiersprachen (SCHNEIDER 74, BRENDL 77, GÜTLER 77) ermöglicht die Einbeziehung kontextabhängiger Eigenschaften in die syntaktische Definition sowie die formale Definition der Semantik.

Ursprünglich entstanden Graph-Grammatiken zur Beschreibung zweidimensionaler Objekte, wie sie bei der Mustererkennung auftreten (BUNKE 83). Die Spezifikation hochintegrierter Schaltkreise kann deshalb mit Hilfe attributierter Graphen auf sehr naheliegende Weise erfolgen. Die Knoten und Kanten des Graphen repräsentieren die primitiven Elemente des Schaltkreises und ihre Struktur, während die funktionalen und zeitlichen Eigenschaften durch Attribute der Knoten definiert werden.

Da ein komplexer Entwurf aus mehreren Zellhierarchien besteht, die untereinander verbunden sind, werden im abstrakten Modell quasihierarchische Graphen verwendet (GÜTLER 77). Dies ermöglicht die Beibehaltung des formalen Beschreibungsmittels bei einer Erweiterung des CERES Systems um Sprachelemente zur modularen, hierarchischen Spezifikation.

Die Darstellung und Definition der Produktionen ist an (BRENDDEL 79) angelehnt, die Erweiterung der Graph-Grammatiken um Attribute ist in (BUNKE 83) und (GÜTLER 83) beschrieben. Für eine effiziente Implementierung des Synthese-Systems ist die Realisierung der Syntaxanalyse und die dafür bekannten Verfahren von großer Bedeutung. Der letzte Abschnitt enthält deshalb einige Aspekte und Ergebnisse, die bei einer Realisierung des Synthese-Systems Ausgangspunkt einer Implementierung sein könnten.

3.1. Quasihierarchische Graphen

Sowohl die funktionale Spezifikation als auch die Beschreibung des Layouts eines integrierten Schaltkreises ist logisch in gewisse Teilkomplexe strukturiert. Da die Modellbildung diese Struktur wiedergeben soll, ist es notwendig, im verwendeten Formalismus eine Hierarchiebildung zu ermöglichen. Die Verwendung hierarchischer Graphen gestattet eine formale Beschreibung auf verschiedenen Abstraktionsebenen, wobei ein Knoten eines hierarchischen Graphen wiederum einen Graphen enthalten kann. Jede Kante hat ihren Anfangs- und Endknoten im gleichen Graphen. Es ist nicht möglich, Kanten von einem beliebigen Knoten auf Knoten in anderen Graphen verweisen zu lassen.

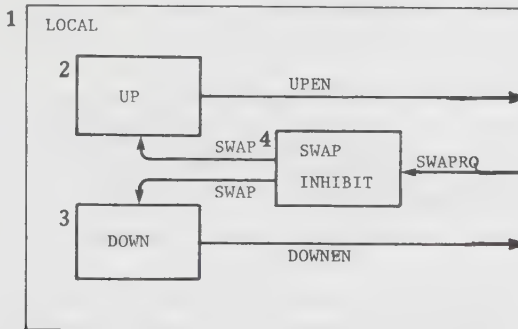


Bild 3.1: Lokale Steuerlogik als quasihierarchischer Graph

Bei der Beschreibung der Struktur integrierter Schaltkreise werden

Kanten zwischen den Zellen verwendet, welche die gegenseitige geometrische Anordnung beschreiben. Dies entspricht der Definition hierarchischer Graphen. Die physikalischen Verbindungen der primitiven Elemente sind dagegen durch Kanten definiert, die eine direkte Verbindung zwischen diesen Elementen, unabhängig von der Einbettung der Knoten in andere Graphen, herstellen. Diese weniger strenge Form der Modularisierung erfordert die Erweiterung des Formalismus auf quasihierarchische Graphen. Bild 3.1 zeigt die lokale Steuerlogik des SORT Bausteins als quasihierarchischen Graphen in impliziter Darstellung (GÜTLER 78). Die explizite Darstellung, bei der das Enthaltensein eines Graphen in einem Knoten der nächsthöheren Hierarchiestufe durch 'in' Kanten dargestellt wird, ist zwar enger an die formale Definition angelehnt, aber graphisch nicht so einprägsam (siehe Bild 3.2).

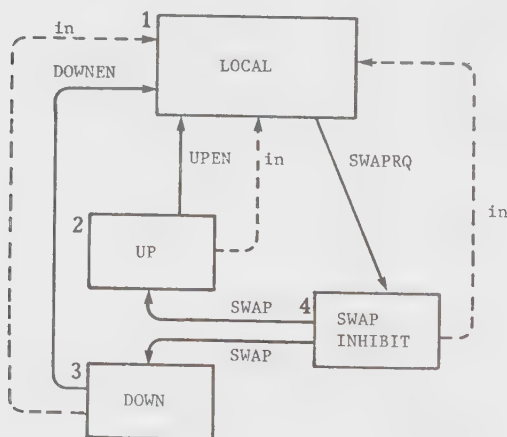


Bild 3.2: Explizite Notation von Bild 3.1

Definition 3.1: Quasihierarchischer Graph:

Sei $NODES$ eine endliche Menge, NL die Menge der Knotenmarkierungen, EL die Menge der Kantenmarkierungen, dann ist $G=(NODES,EDGES)$ ein bewerteter, gerichteter, quasihierarchischer Graph mit:

- o $LABEL : NODES \rightarrow NL$ ist die Knotenmarkierungsfunktion
- o $EDGES \subseteq NODES \times NODES \times (EL \cup \{in\})$ ist die markierte Kantenmenge,
- o $(\forall a,b,c \in NODES): (a,b,in) \wedge (a,c,in) \in EDGES \rightarrow b=c;$
der Graph besitzt bezüglich der in-Kanten Waldstruktur

Da im folgenden nur bewertete, gerichtete und quasihierarchische Graphen betrachtet werden, sprechen wir allgemein von Graphen. Knoten, die nicht auf der untersten Hierarchiestufe stehen und weitere Graphen enthalten, sind im Gegensatz zu (GÖTTLER 77) ebenfalls mit einem Bezeichner markiert. Dies gestattet uns, die in der funktionalen oder technologischen Beschreibung verwendeten Namen auch im Graphen zusätzlich zur inneren Struktur zu verwenden. Die in der Definition der Knotenmenge zur Unterscheidung der Knoten verwendete Menge M wird gewöhnlich als Numerierung der Knoten angegeben. Da diese nur zur Angabe der Attribute benötigt werden, schreiben wir sie außerhalb des Knotens, die Markierung in den Knoten.

Definition 3.2: Attributierter Graph:

Sei A eine Menge von Attributen, deren Elemente Funktionen sind, die jedem Knoten einen Attributwert zuordnen. Ein attributierter Graph $AG=(NODES,EDGES,ATT)$ ist ein Tripel mit:

- o $G = (NODES,EDGES)$ ist ein quasihierarchischer Graph
- o $ATT : NODES \rightarrow Potenzmenge(A)$ ist die Attributierungsfunktion, die jedem Knoten eine Menge von Attributen zuordnet.

Da die semantische Information bei der Beschreibung hochintegrierter Schaltkreise den Knoten der Graphen zugeordnet wird, kann auf eine Attributierung der Kanten (BUNKE 83) verzichtet werden. Ein einfaches Beispiel für Knotenattribute ist die Signallaufzeit durch ein Schaltnetz. In Bild 3.3 sind diese Verzögerungszeiten an den entsprechenden Knoten dargestellt.

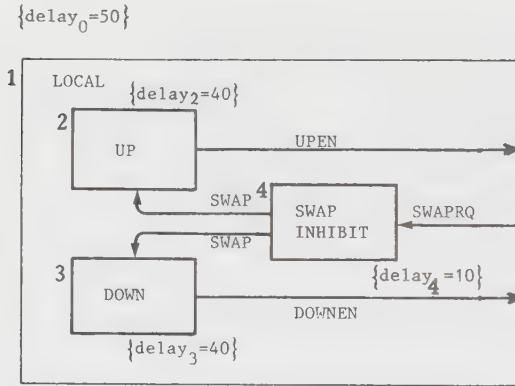


Bild 3.3: Zeitattribute für die lokale Steuerlogik

3.2. Ersetzungsregeln auf Graphen

Das Ersetzungsprinzip der Chomsky-Grammatiken für Zeichenketten wurde in der Theorie der Graph-Grammatiken auf mehrdimensionale Zeichensysteme erweitert. Eine Einführung in Theorie, Anwendung und Implementierung enthält (NAGL 79), zahlreiche Veröffentlichungen und eine Bibliographie (EHRIG 83).

Das Prinzip der Graphersetzungssysteme ist eine Erweiterung des Produktionsbegriffs bei Zeichenketten-Grammatiken. Während bei Zeichenketten durch Anwendung der Produktion ein Zeichen oder ein Teil der Zeichenkette (die linke Seite der Produktion) durch eine neue Zeichenkette (die rechte Seite der Produktion) ersetzt wird, muß bei Graph-Ersetzungssystemen eine im allgemeinen komplexere Struktur aus dem Wirtsgraphen herausgelöst und durch einen neuen Graphen ersetzt werden. Während das Einfügen neuer Zeichenketten an Stelle der ersetzten problemlos durchgeführt werden kann, ist im Fall der Graphersetzung eine Einbettung durchzuführen. Dazu werden zwischen dem, nach Entfernen der linken Seite entstehenden Restgraphen und der neu einzubettenden rechten Seite mit Hilfe der Einbettungsüberführung neue Kanten generiert.

Die Methode mit der diese Einbettungsüberführung definiert wird, unterscheidet sich bei verschiedenen Ansätzen:

- o Beim kategoriellen Ansatz (EHRIG 73, ROSEN 75, SCHNEIDER 77) wird die Einbettungsüberführung durch einen Pushout-Konstrukt definiert.
- o Beim operationellen Ansatz (PFALTZ 69, NAGL 74) definieren Operatoren ausgehend von Knoten der linken Seite, wie die neuen Knoten der rechten Seite mit dem Restgraphen verbunden werden.
- o Der graphentheoretische Ansatz (GÖTTLER 77, BRENDEL 79) ermöglicht eine Spezifikation der Einbettungsüberführung durch Angabe von Kanten, welche die Einbettung der rechten Seite in den Kontext des Restgraphen angeben.

Für die formale Beschreibung des funktionalen Entwurfs und der Struktur hochintegrierter Schaltkreise wird eine, gegenüber der allgemeinen Form stark eingeschränkte, Klasse von Graph-Ersetzungssystemen benötigt. Wie bei anderen Anwendungen (BRENDDEL 79, SCHNEIDER 74, GÖTTLER 77) dienen die Einschränkungen einer vereinfachten Definition der Ersetzungsregeln, die eine verständlichere Darstellung des Problems gestatten. Die wesentliche Einschränkung des graphentheoretischen Ansatzes ist, daß in den Produktionen nur Einbettungen in den Restgraphen angegeben werden können, für die vor Anwendung der Produktion bereits eine Kante von einem Knoten der linken Seite in den Kontext existierte. Insbesondere können nicht beliebig weit entfernte Knoten des Restgraphen mit Knoten der rechten Seite verbunden werden. Für die praktische Anwendung bedeutet dies keine wesentliche Einschränkung. Die explizite graphische Darstellung von Produktionen und Einbettungsüberführung ermöglicht eine leicht verständliche Definition des Produktionsbegriffes.

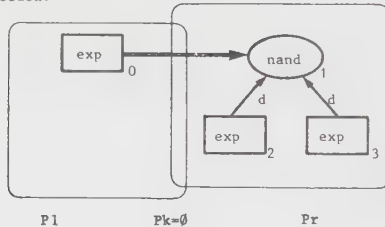
Definition 3.3: Graph-Produktion:

Die Graph-Produktion $P=(Pl,Pr,e,Ap)$ definiert die Ersetzung eines Teilgraphen, die Einbettung des neuen Teils und die Berechnung der Attribute

- o Die nichtattributierten Graphen $Pl=(Nl,El)$ und $Pr=(Nr,Er)$ bilden die linke und rechte Seite der Produktion
- o Der linker und rechter Seite gemeinsame Teilgraph $P_k = Pl \cap Pr$ dient zur Beschreibung des unveränderten Kontextes

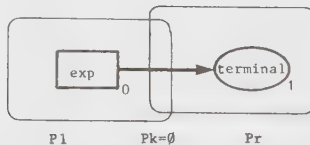
- o Die Kanten der Einbettungsüberführung $e \in (N_l/N_k) \times (N_r/N_k)$ werden durch Doppelstriche dargestellt.
- o A_p ist eine endliche Menge partieller Funktionen $F: (N_l \cup N_r) \rightarrow A$ die den Knoten der Produktion Attributwerte $a \in A$ zuordnen

expression:



$$\begin{aligned} \text{delay}(0, \text{exp}) &\leftarrow \text{delay}(1, \text{nand}) + \max(\text{delay}(2, \text{exp}), \text{delay}(3, \text{exp})). \\ \text{delay}(1, \text{nand}) &\leftarrow 6. \end{aligned}$$

terminal:



$$\begin{aligned} \text{delay}(0, \text{exp}) &\leftarrow \text{delay}(1, \text{terminal}). \\ \text{delay}(1, \text{terminal}) &\leftarrow 0. \end{aligned}$$

Bild 3.4: Notation von Graph-Produktionen

Im Gegensatz zu (GÖTTLER 77) werden in der obigen Definition die Kontextknoten als der linken und rechten Seite gemeinsam betrachtet. Dies führt bei der Definition der Ableitung dazu, daß sowohl die Knoten des Kontexts, als auch die Kanten der linken Seite, welche Kontextknoten tangieren, im Wirtsgraphen vorhanden sein müssen, ehe eine Ersetzung ausgeführt werden kann. Dadurch ist eine einfachere Formulierung der Produktionen mit Prüfung der Umgebung möglich. Die Kanten der Einbettungsüberführung geben an, auf welchen Knoten der rechten Seite die in einen Knoten der linken Seite führenden oder aus ihm herausgehenden Kanten übertragen werden sollen. Eine Auswahl von Kanten, abhängig von deren Markierung, eine Umorientierung oder Ummarkierung der Kanten

erfolgt im Gegensatz zu (NAGL 74) und (SCHNEIDER 77) nicht.

Die in Bild 3.4 angegebenen Produktionen erzeugen einen binären Baum von NAND-Gattern. Da die Ersetzung vom Kontext unabhängig ist, genügen kontextfreie Produktionen zur Beschreibung der Struktur. Während die 'expression' Produktion die Baumstruktur erzeugt, ordnet die 'terminal' Produktion den Gattern entsprechende Eingangssignale zu. 'Ausgehend von einem 'exp' markierten Knoten wird die Generierung des Baums durch sukzessive Ersetzung der 'exp'-Knoten durch 'nand'-Knoten mit zwei neuen 'exp'-Knoten als Eingängen durchgeführt. Dem neuen 'nand'-Knoten wird dabei die vom 'exp'-Knoten der linken Seite ausgehende 'd'-Kante durch die Einbettungsüberführung übertragen. Der Generierungsvorgang bricht ab, wenn alle 'exp'-Knoten durch 'nand'-Knoten ersetzt wurden.

Die Attributfunktion berechnet die Signallaufzeit an den erzeugten Knoten in Abhängigkeit von den Verzögerungszeiten der Eingangssignale, indem ausgehend von den Blättern des Baumes die Attribute der Wurzel berechnet werden. Die Attributfunktion berechnet für jeden 'nand'-Knoten eine Verzögerung von sechs Zeiteinheiten, die zur maximalen Verzögerung der beiden Teilbäume addiert wird.

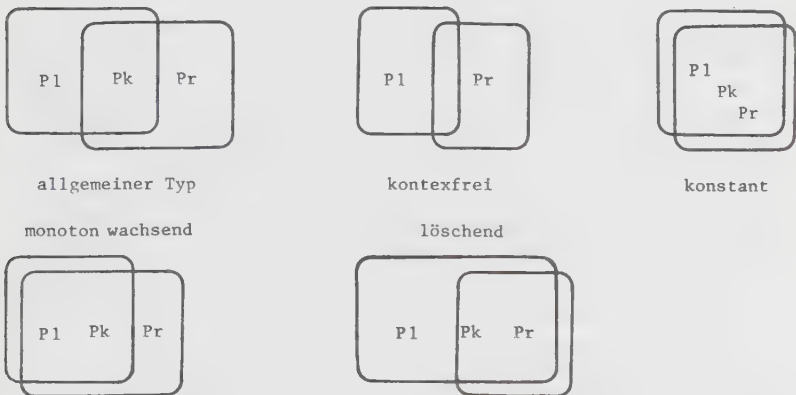


Bild 3.5: Produktionentypen

Die graphische Notation von Graph-Produktionen mit überlappender linker und rechter Seite erlaubt eine einfache Unterscheidung verschiedener Produktionstypen nach allgemeinen kontextsensitiven, kontextfreien, konstanten, monoton wachsenden sowie löschenden Produktionen (siehe Bild 3.5). Bei kontextfreien Produktionen in der hier verwendeten Definition besteht die linke Seite im Gegensatz zur Definition in (NAGL 79) im allgemeinen aus mehr als einem Knoten. Unabhängig von der Ersetzung der Knoten können jedoch stets Kanten gelöscht, erzeugt oder ersetzt werden.

3.3. Ableitung und programmierte Graph-Grammatik

Der Begriff der Ableitung wird wie bei Zeichenketten-Grammatiken als eine Folge von Ersetzungen von linken Seiten der Produktionen im Wirtsgraphen durch die entsprechenden rechten Seiten definiert. Zusätzlich werden die durch die Einbettungsüberführung definierten Kanten generiert. Die Attribute der neuen Knoten der rechten Seite werden aufgrund der Attributierungsfunktionen berechnet.

Definition 3.4: Direkte Ableitung:

Sei EL die Menge der Kantenmarkierungen, $Pl=(Nl,El)$ und $Pr=(Nr,Er)$ die linke und rechte Seite der Produktion $P=(Pl,Pr,e,Ap)$. Ein attributierter Graph $AG'=(N',E',A')$ ist aus $AG=(N,E,A)$ mittels der Produktion $P=(Pl,Pr,e,Ap)$ direkt ableitbar wenn gilt:

- o $Pl \subseteq G, Pr \subseteq G'$ Die Produktion ist anwendbar, wenn die linke Seite Pl Teilgraph von G ist; Pl wird in G durch Pr ersetzt.
- o $G/Pl = G'/Pr$ der Restgraph bleibt unverändert,
- o $\forall k \in (E/El), m \in EL \cup \{in\}: (k,kl,m) \in E \wedge (kl,kr) \in e \rightarrow (k,kr,m) \in E'$,
die Kanten zu Knoten der linken Seite werden durch Einbettungskanten auf entsprechende Knoten der rechten Seite übertragen
- o $\forall k \in (E/El), \forall kk \in (El \cap Er), \forall m \in EL:$
 $(k,kk,m) \in E \rightarrow (k,kk,m) \in E', (kk,k,m) \in E \rightarrow (kk,k,m) \in E'$
 Die Verbindungskanten des Kontextteils der Produktion zum Wirtsgraphen bleiben bei der Ersetzung unverändert erhalten.
- o Den Knoten der Produktion werden durch die Attributierungsfunktionen Attributwerte zugeordnet.

$$\text{delay}(G) = \text{delay}(O, \text{nand}) + \max(\text{delay}(1, \text{exp}), \text{delay}(2, \text{exp})) .$$

$$\begin{aligned} \text{delay}(G') = & \text{delay}(O, \text{nand}) + \\ & \max(\text{delay}(1, \text{exp}), \\ & (\text{delay}(2, \text{nand}) + \max(\\ & \text{delay}(3, \text{exp}), \text{delay}(4, \text{exp})))). \end{aligned}$$

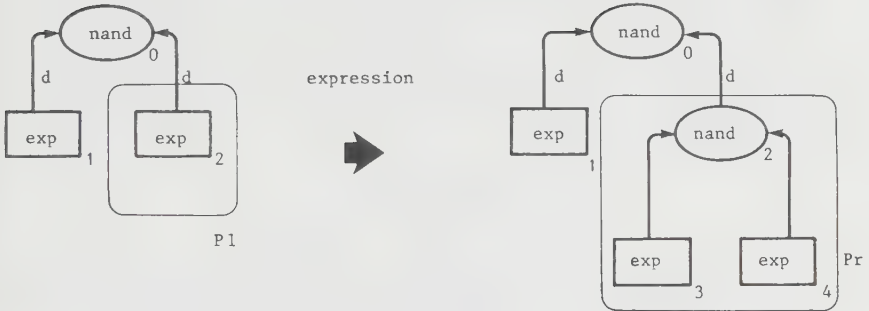


Bild 3.6: Anwendung einer Produktion

Eine Produktion ist auf den Wirtsgraphen G genau dann anwendbar, wenn alle Knoten und Kanten der linken Seite $P1$ der Produktion in diesem enthalten sind. Zusätzliche Kanten zwischen Knoten des Wirtsgraphen, die Knoten der linken Seite entsprechen, können vorhanden sein (Definition Teilgraph siehe NAGL 79). Sie werden durch die Anwendung der Produktion jedoch gelöscht, wenn nicht Anfangs- und Endknoten der Kanten vollständig im Kontextteil der Produktion liegen. Die Anwendung der Produktion aus Bild 3.4 auf einen Graphen ist in Bild 3.6 dargestellt. Dabei werden den neu eingefügten Knoten Attribute zugeordnet. Im angegebenen Beispiel der Verzögerungszeiten kann eine Auswertung jedoch erst erfolgen, wenn der Ableitungsvorgang beendet ist und die Attribute der Blätter des Baumes bekannt sind. Diese 'bottom up' Auswertung der Attribute ist nur bei der ausschließlichen Verwendung abgeleiteter Attribute möglich. Bei allgemeinen Abhängigkeiten der Attribute sind aufwendigere Verfahren notwendig (WILHELM 79).

Da bei allgemeinen Graph-Grammatiken die Reihenfolge, mit der Produktionen auf einen Graphen angewendet werden, nichtdeterministisch ist, hat es sich bei der Beschreibung von Transformationen als zweckmäßig erwiesen, die Reihenfolge der Produktionsanwendung zu beschränken (SCHNEIDER 74, NAGL 79). Durch Angabe von Kontrollgraphen ist es möglich,

abhängig von der Anwendbarkeit einer Produktion, Verzweigungen, Sequenzen und Wiederholungen von Produktionen zu formulieren. Die generative Mächtigkeit dieser programmierten Graph-Grammatiken entspricht zwar der der allgemeinen Graph-Grammatiken, die Formulierung der Produktionen ist im allgemeinen jedoch einfacher, da zur Erzwingung bestimmter Reihenfolgen keine nichtterminalen Knoten und Kanten erzeugt werden müssen.

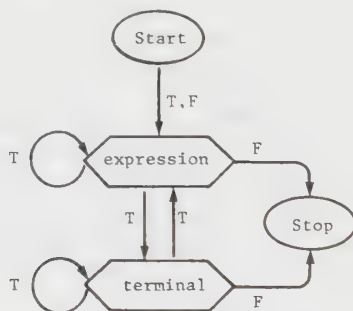


Bild 3.7: Beispiel eines Kontrollgraphen für Ableitungsfolgen

Definition 3.5: Kontrollgraph:

Ein Kontrollgraph $KG=(KN,KE)$ ist ein Graph mit:

- o $KN = P \cup \{\text{Start}, \text{Stop}\}$ Die Knotenmenge besteht aus einer endlichen Menge von Produktionen sowie einem 'Start' und einem 'Stop' Knoten.
- o $KE \subseteq KN/\{\text{Stop}\} \times KN/\{\text{Start}\} \times \{T, F\}$
Die Kanten sind mit T und F markiert. Keine Kante führt in den 'Start' und keine aus dem 'Stop' Knoten.

Der Kontrollgraph definiert die Reihenfolge der Produktionen, die auf einen attribuierten Graphen angewendet werden. Beginn ist der 'Start'-Knoten. Ist man bei der Abarbeitung des Kontrollgraphen bei einem mit einer Produktion markierten Knoten angelangt, wird der weitere Ableitungsvorgang durch die Anwendbarkeit dieser Produktion bestimmt. Ist sie anwendbar (die linke Seite ist Teilgraph des Wirtsgraphen), dann wird

eine der Produktionen als nächste angewendet, die über die T-Kante erreichbar sind, sonst eine durch F-Kanten bestimmte. Da in der Definition des Kontrollgraphen aus jedem Knoten mehrere Kanten herausführen können, ist der Pfad durch den Kontrollgraphen im allgemeinen nichtdeterministisch. Die Stelle im Wirtsgraphen an der die Produktion angewendet wird, ist ebenfalls nicht eindeutig bestimmt, da die linke Seite der Produktion mehrfach vorkommen kann. Der Ableitungsvorgang ist beendet, wenn der 'Stop' Knoten erreicht wird. Der in Bild 3.7 angegebene Kontrollgraph enthält Sequenzen, Wiederholungen und Verzweigungen und erzeugt mit Hilfe der obigen Produktionen einen Baum von NAND-Gattern, der an den Blättern ausschließlich 'terminal'-Knoten enthält.

Definition 3.6: Programmierte Graph-Grammatik:

Eine programmierte Graph-Grammatik $GG=(NL,EL,Ap,Go,P,KG)$ besteht aus:

- o NL und EL sind die Knoten- und Kantenmarkierungsalphabete
- o Ap ist die endliche Menge der Knotenattribute
- o Go ist der Startgraph
- o P ist eine endliche Menge von Produktionen
- o KG ist der die Ableitung steuernde Kontrollgraph

Abweichend von der üblichen Definition einer Grammatik wurde auf die Unterscheidung zwischen terminalen und nichtterminalen Markierungen bei Knoten und Kanten verzichtet. Die nichtterminalen Symbole dienen üblicherweise als vorübergehend benutzte Markierungen, die in der durch die Grammatik erzeugten Sprache nicht mehr auftreten. Sie sind ein einfaches Hilfsmittel um das Ende des Ableitungsprozesses zu definieren. Da durch die Einführung des Kontrollgraphen die Reihenfolge der Anwendungen der Produktionen definiert ist und das Ende des Ableitungsprozesses eintritt, wenn im Kontrollgraph der 'Stop' Knoten erreicht wird, ist die Einführung nichtterminaler Markierungen entbehrlich.

Definition 3.7: Sprache einer Graph-Grammatik:

Die Menge der durch eine programmierte Graph-Grammatik erzeugten Graphen wird Sprache $L(GG)$ der Grammatik GG genannt und folgendermaßen erzeugt:

- o Beginne mit dem Startgraphen der Grammatik und dem Startknoten des Kontrollgraphen
- o Wende die Produktionen der Grammatik in einer Reihenfolge an, die der Kontrollgraph zuläßt.
- o Wird der 'Stop' Knoten des Kontrollgraphen erreicht, ist der Ableitungsprozess beendet.
- o Jeder so erzeugte Graph ist Element von $L(GG)$

3.4. Syntaxanalyse und Implementierungsaspekte

Mit den obigen Definitionen ist es nun möglich, sowohl die Menge aller durch eine funktionale Beschreibung spezifizierbaren Entwürfe, als auch die mit einer Zellbibliothek erzeugbaren integrierten Schaltungen formal zu erfassen. Da die Synthese auf die Syntaxanalyse bei Graph-Grammatiken zurückgeführt werden soll (siehe Abschnitt 1.3.1), wird in diesem Abschnitt darauf noch näher eingegangen.

Bei der Ableitung von Zeichenketten erhält man einen Ableitungsbaum, wenn man alle Ersetzungsschritte in einer Darstellung festhält. Da bei Graph-Grammatiken die lineare Struktur der Zeichenketten nicht mehr vorliegt, erhält man einen Ableitungsgraphen, der die Ableitung charakterisiert. Diesen Ableitungsgraphen kann man konstruieren, indem man beginnend mit dem Startgraphen alle Ersetzungen unter die entsprechenden Knoten zeichnet. Zwischen den ersetzten Knoten der linken Seite der Produktion und den Knoten der rechten Seite werden Substitutionskanten gezogen. Da durch die Einbettungsüberführung stets neue Kanten zwischen der eingesetzten rechten Seite und dem Restgraphen generiert werden, erhält man im allgemeinen keinen Ableitungsbaum, sondern einen Graphen.

Wesentlich bei der Ableitung ist die Generierung der Attribute für die Knoten der rechten Seite. Wie bei Zeichenketten kann zwischen abgeleiteten und ererbten Attributen unterschieden werden (WILHELM 79). Die

Wohldefiniertheit und Eindeutigkeit der Attribute und ihrer Auswertungsfunktionen muß dabei wie bei attribuierten Zeichenketten-Grammatiken gewährleistet sein (GÖTLER 83). Der in Bild 3.8 dargestellte Ableitungsgraph zeigt die Erzeugung der Vergleicherschaltung des SORT Bausteines (siehe Abschnitt 2.3.1) mit Hilfe der obigen Produktionen und des Kontrollgraphen aus Bild 3.7. Wegen der einfachen Struktur der kontextfreien Produktionen hat der Ableitungsgraph Baumstruktur. Die Knotenattribute in geschweiften Klammern geben die Signallaufzeit an diesen Knoten bei einer angenommenen Gatterverzögerung von sechs Zeiteinheiten an. Eine ausführliche Beschreibung der Attributberechnung enthält (GÖTLER 83).

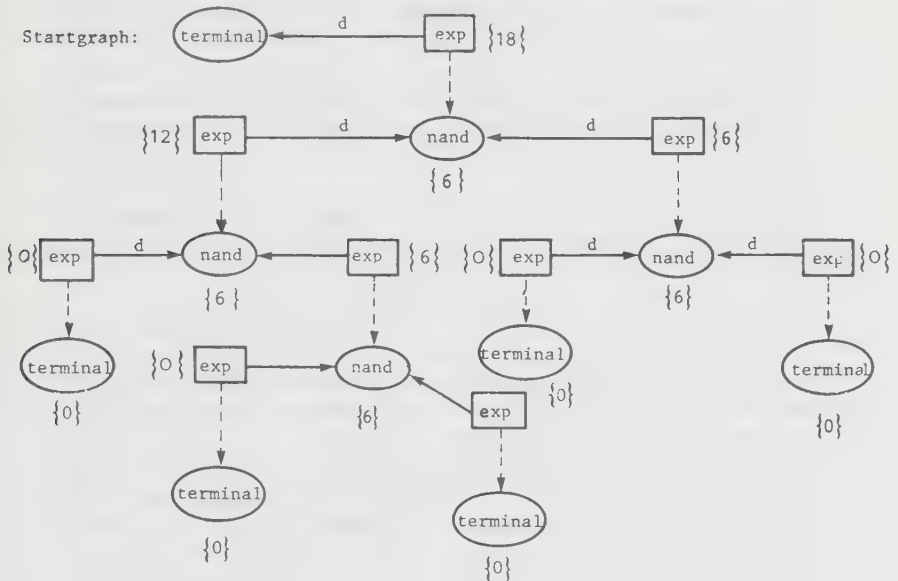


Bild 3.8: Ableitungsgraph mit Attributauswertung

Effiziente Syntaxanalyseverfahren sind, wie bereits in Abschnitt 1.3.1 erwähnt, nur für eine stark eingeschränkte Klasse von Graph-Grammatiken bekannt. Dies sind die Präzedenz-Graph-Grammatiken, die lediglich kontextfreie Ersetzungen mit einer eingeschränkten Einbettungsüberführung zulassen (KAUL 83). Da für den hier verfolgten Anwendungsfall die

Produktionen ganz wesentlich die Umgebung der Ersetzung berücksichtigen, ist diese Graph-Klasse nicht ausreichend. Prinzipiell ist das Wortproblem für jede monotone Graph-Grammatik entscheidbar (NAGL 79), die Syntaxanalysealgorithmen sind dann Backtracking-Verfahren, die nur sehr ineffizient implementiert werden können.

Die Einschränkung der bei einem Ableitungsschritt möglichen Produktionen durch die Einführung programmierter Graph-Grammatiken vereinfacht aber nicht nur die Definition der Grammatik, sondern ermöglicht eine effiziente Implementierung des Syntaxanalyseverfahrens. Im Gegensatz zur Ableitung beginnt man bei der Syntaxanalyse programmierter Graph-Grammatiken mit dem abgeleiteten Graphen und dem 'Stop' Knoten des Kontrollgraphen. Die Reduktion des Graphen erfolgt durch Umkehrung der vom 'Stop' Knoten aus erreichbaren Produktionen und die Ersetzung der rechten Seite durch die linke. Dieses Verfahren ist zwar ebenfalls nichtdeterministisch, durch die Einschränkung der in Frage kommenden Produktionen ist jedoch auch für große Graphen eine Syntaxanalyse durchführbar (BARTSCH 83).

Eine Übersicht der verschiedenen Implementierungen von Graph-Grammatiken enthält (NAGL 79). Die Zielsetzung dieser Arbeiten ist kein Einsatz der Implementierung für ein bestimmtes Anwendungsgebiet, sondern die Erprobung verschiedener Konzepte zur Speicherung von Graphen und zur Realisierung der Ableitung. So wurde in (BRENDDEL 76) die allgemeinste Form von Einbettungs- und Verbindungsüberführungen für Graph-Grammatiken ohne Beschränkung der Größe von Produktionen und Graphen realisiert. Auf dieser Arbeit aufbauend wird zur Zeit eine Erweiterung für attributierte Graph-Grammatiken erstellt. Diese Implementierungen könnten als Basis für die Erstellung eines Prototyps des Synthese-Systems dienen. Die dabei gesammelten Erfahrungen würden dann eine effiziente Implementierung ermöglichen, die auf die Bedürfnisse dieser speziellen Anwendung zugeschnitten ist.

An important approach to cope with design cost of complex integrated circuits is the synthesis of highly regular structures.

R.W. Hartenstein

4. SPEZIFIKATION UND SYNTHESE DES DATENPFADES

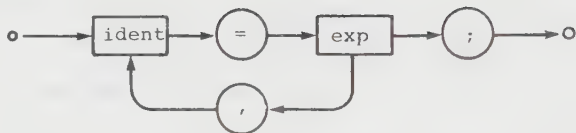
Ausgehend von der funktionalen Spezifikation des Datenpfades und der formalen Definition der Syntax der Entwurfssprache werden in diesem Kapitel Methoden entwickelt, die es erlauben, zu einer gegebenen Spezifikation mehrere Realisierungsvarianten zu überprüfen. Grundlegend für die eindeutige Zuordnung zwischen Spezifikation und Realisierung ist eine Entwurfssprache, deren Syntax und Semantik sich an den Elementen orientiert, die in der Realisierung als VLSI-Schaltkreis verwendet werden. Änderungen der Spezifikation müssen für den Entwickler nachvollziehbare Auswirkungen in der technologischen Realisierung haben. Dies bedeutet, daß jedes Sprachelement der Entwurfssprache eine eindeutige Interpretation auf der Schaltkreisebene haben muß. Da diese im allgemeinen technologieabhängig ist, wird im folgenden eine NMOS-Technologie mit den in Abschnitt 1.2.5 eingeführten Entwurfsregeln zugrunde gelegt.

4.1. VERKNÜPFUNG LOGISCHER SIGNALE

Das in Bild 4.1 dargestellte Syntaxdiagramm für die Deklaration der Verknüpfung logischer Signale ('terminal') orientiert sich an ERES (GARDILL 77, BRENDL 82), wobei nur Operatoren zugelassen sind, die eine direkte Entsprechung in der NMOS-Technologie aufweisen. Für Technologien, welche die Bildung logischer Ausdrücke mit anderen Operatoren erlauben, sind sowohl die Syntaxdiagramme, als auch die Graph-Grammatiken entsprechend zu modifizieren.

Bei der hier verwendeten Notation der Syntaxdiagramme beginnen und enden Kanten stets in Knoten. Dies ist eine Variante der in (WIRTH 76) eingeführten Syntaxdiagramme, bei denen Kanten auch auf andere Kanten führen können, was bei Verzweigungen zu Interpretationsschwierigkeiten führen kann.

terminal:



expression:

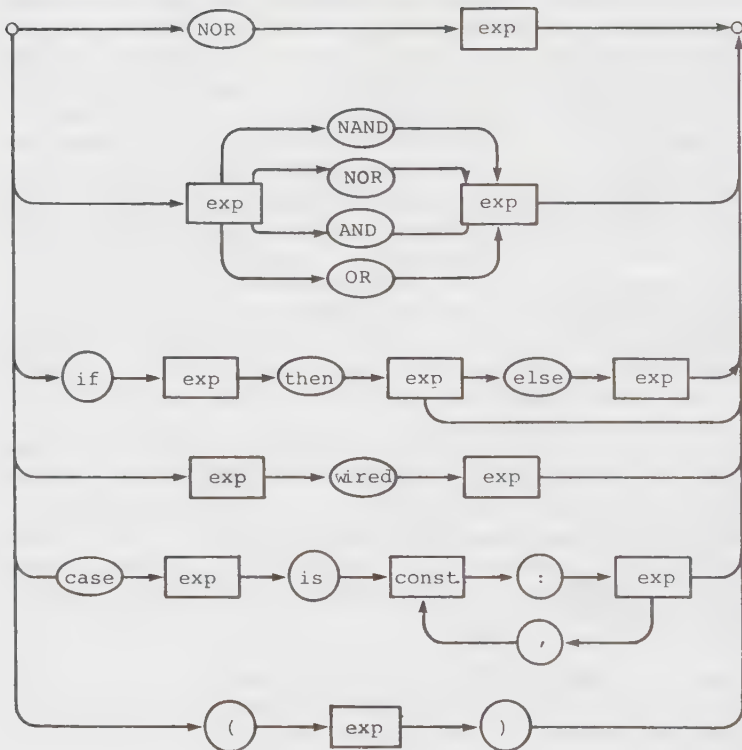


Bild 4.1: Syntaxdiagramm Deklaration logischer Verknüpfungen

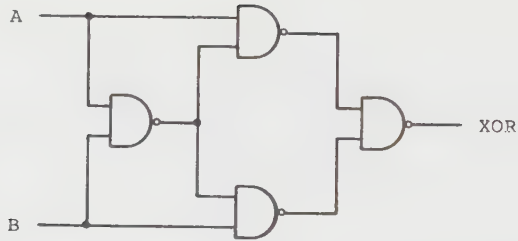
4.1.1. Elementare Operatoren

Die Grundelemente des Schaltungsentwurfs in NMOS-Technik sind NOT, NOR und NAND Gatter. Alle anderen Funktionen mit Ausnahme der TransfERGatter (Passtransistor) sind aus diesen zusammengesetzt. Bei Verwendung der NMOS-Technologie können jedoch darüber hinaus innerhalb eines Gatters durch Serien- und Parallelschaltungen von Transistoren AND und OR Verknüpfungen realisiert werden. Da die Spezifikationssprache von realisierungsspezifischen Details unabhängig bleiben soll, sind bei der Deklaration von Terminals diese logischen Verknüpfungen ebenfalls zugelassen, ohne ihre Verwendung in der funktionalen Spezifikation direkt einzuschränken. Eine Überprüfung ob die angebene Schaltung in einer gegebenen Zellbibliothek synthetisierbar ist, erfolgt erst bei der Syntaxanalyse des Entwurfsgraphen mit der Grammatik der Zellbibliothek.

Im Gegensatz zu Ausdrücken in Programmiersprachen haben logische Verknüpfungen in Schaltungen im allgemeinen keine Baumstruktur. Bei synchronen Schaltkreisen ohne Rückkopplung entstehen vielmehr gerichtete Graphen. Im Syntaxdiagramm der Terminaldeklaration kommt dies dadurch zum Ausdruck, daß zum Bezeichner des Terminals weitere lokale Bezeichner deklariert werden können, deren Gültigkeitsbereich auf die Deklaration beschränkt ist. Sie dienen lediglich zur Spezifikation von Teilausdrücken, die an mehreren Stellen verwendet werden. Im Schaltbild des Beispiels 4.1 kommt dies durch mehrfache Kanten zum Ausdruck, die von einem Gatter zu verschiedenen anderen führen, in denen der Teilausdruck verwendet wird. Die anderen in der Deklaration auftretenden Bezeichner sind Eingangsterminals, die an anderer Stelle zu deklarieren sind.

Die Struktur des die funktionale Spezifikation repräsentierenden Entwurfsgraphen wird durch eine Graph-Grammatik definiert. Da dieser nicht den Beschränkungen einer linearen Notation unterliegt, kann die Verwendung gemeinsamer Teilausdrücke durch die Zusammenführung verschiedener Quellen in Produktion L4 dargestellt werden (siehe Bild 4.2). Die in der linearen Notation auftretenden lokalen Hilfsvariablen sind dabei nicht notwendig. Die angegebenen Produktionen werden bei der Einführung weiterer Operationen in den folgenden Abschnitten entsprechend ergänzt.

Da die funktionale Spezifikation keine Einschränkungen bezüglich der Verwendung der verschiedenen Operatoren aufweisen soll, enthält der zur Grammatik gehörende Kontrollgraph keine Beschränkungen bezüglich der



$$\begin{aligned} \text{terminal } \text{XOR} &= (\text{A nand X}) \text{ nand } (\text{X nand B}), \\ \text{X} &= \text{A nand B}; \end{aligned}$$

Beispiel 4.1: XOR Schaltung mit gemeinsamen Teilausdruck

Reihenfolge, in der die Produktionen angewendet werden können und ist deshalb hier nicht explizit angegeben.

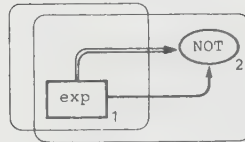
Die funktionale Spezifikation soll ein exaktes Abbild der synthetisierten Schaltung darstellen, weshalb die Operatoren direkt auf entsprechende Elemente der Technologie abbildbar sein müssen. Bei der hier behandelten NMOS-Technologie sind jedoch AND und OR nur in bestimmten Kontexten zulässig. Die Grammatik der Zellbibliothek wird deshalb gegenüber der Grammatik der funktionalen Spezifikation einige Einschränkungen aufweisen.

Prinzipiell besteht jedes Gatter der NMOS-Technologie aus einem Verarmungstransistor (depletion load transistor), der als Arbeitswiderstand an die Versorgungsspannung angeschlossen ist. Die logische Funktion wird durch Anreicherungstransistoren (enhancement transistor) realisiert, die sowohl in Reihe, als auch parallel geschaltet werden können. Dadurch entstehen Gatter mit NAND beziehungsweise NOR Funktion. Werden in einem Gatter jedoch Reihen und Parallelschaltungen kombiniert, entstehen Gatter mit komplexerer Funktion, die durch eine übergeordnete NAND- oder NOR-Operation mit einer Kombination von AND- und OR-Operationen dargestellt werden können. Dies soll am Beispiel einer XNOR-Schaltung erläutert werden.

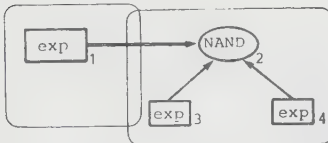
Startgraph



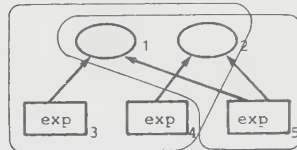
L1: NOT Operation



L2: NAND Operation

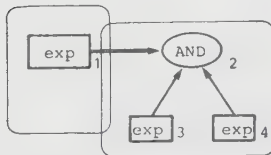


L4: Gemeinsame Teilausdrücke

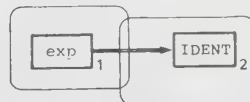


L3: NOR Operation analog L2

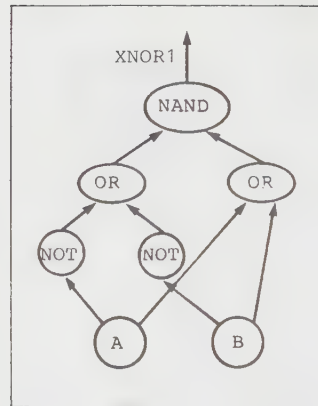
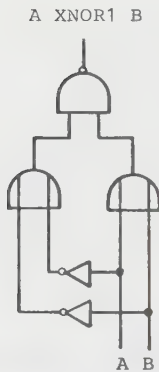
L5: AND Operation



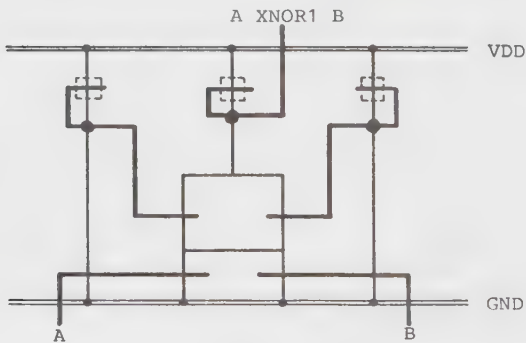
L7: Eingangs-Terminals



L6: OR Operation analog L5

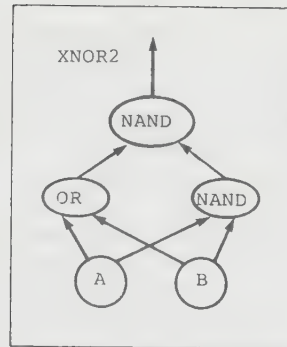
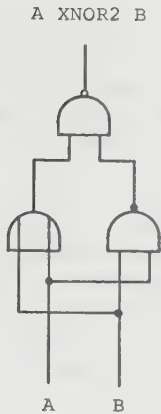


terminal XNOR1 = (not A or not B) nand (A or B)

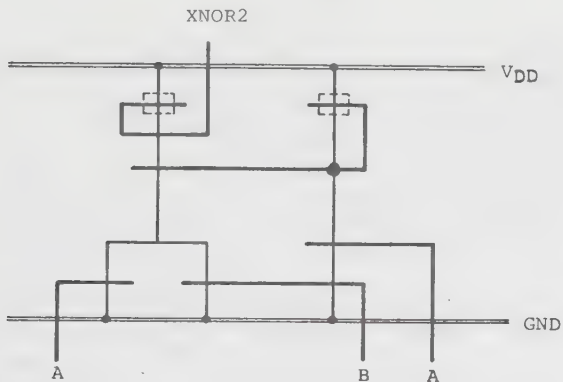


Beispiel 4.2: XNOR Variante 1

Die in der Variante 1 der XNOR-Schaltung auftretenden OR Verknüpfungen können direkt im übergeordneten NAND Gatter realisiert werden. Dadurch entsteht eine Schaltung mit zwei Invertern, bei der die eigentliche XNOR-Operation in einem Gatter realisiert werden kann.



```
terminal XNOR2 = (A nand B) nand (A or B);
```



Beispiel 4.3: XNOR Variante 2

Da bei der Synthese einer Schaltung genau die spezifizierte Struktur entstehen soll, sind die Einschränkungen bei der Verwendung von AND und OR-Operationen in der technologisch orientierten Grammatik der Zellbibliothek zu berücksichtigen. Modifikationen der funktionalen Spezifikation müssen zu gleichartigen Änderungen im synthetisierten Schaltkreis

führen. Dies soll an der Variante 2 der XNOR-Schaltung deutlich gemacht werden, in welcher der erste Term des Ausdrucks direkt durch ein NAND-Gatter ersetzt wurde. Die so entstehende XNOR-Schaltung enthält nur noch zwei Verarmungstransistoren, was eine verlustärmere Realisierung erlaubt.

Zur Synthese einer spezifizierten Schaltung stehen in der Zellbibliothek Elemente für die Serien- und Parallelschaltung zur Realisierung der AND- und OR-Operationen sowie NOT, NAND und NOR zur Verfügung. Die Grammatik, welche die zulässigen Kompositionen dieser Elemente definiert, berücksichtigt die oben erläuterten Einschränkungen. Sie definiert deshalb eine Teilmenge der Grammatik der funktionalen Spezifikation. Bei der Synthese führt dies dazu, daß Schaltkreise, die nicht den obigen Kontextbedingungen entsprechen, Fehlermeldungen hervorrufen (vergl. Abschnitt 1.3.1).

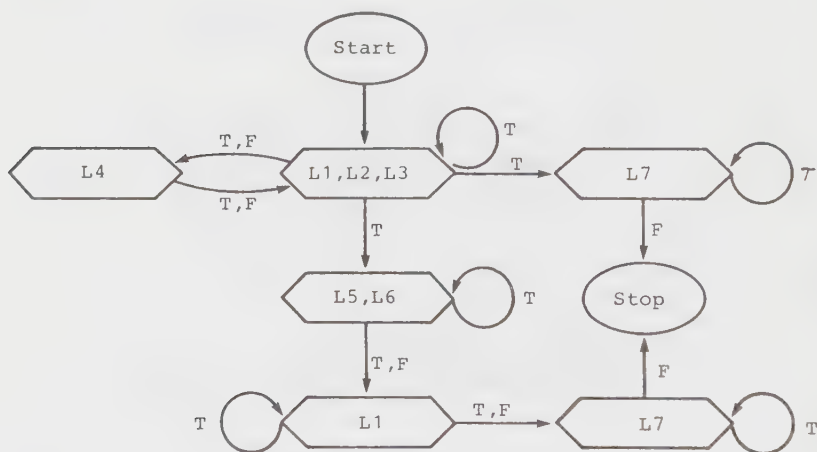


Bild 4.3: Kontrollgraph zur Grammatik in Bild 4.2

Da die Produktionen zur Synthese der logischen Verknüpfung aus Elementen der Zellbibliothek denen der Grammatik der funktionalen Spezifikation entsprechen, unterscheidet sich die Grammatik der Zellbibliothek lediglich durch Einschränkungen der Reihenfolge, in der die Produktionen angewendet werden können. Der in Bild 4.3 angegebene Kontrollgraph

erfordert deshalb die Anwendung zumindest einer NAND/NOR Produktion, ehe eine Serien- Parallelschaltung mit AND/OR Produktionen eingesetzt werden kann. In den dann erzeugten Teilen ist die Verwendung gemeinsamer Teilausdrücke aus elektrischen Gründen nicht mehr möglich. Zur Vereinfachung des Kontrollgraphen sind in den Produktionsknoten teilweise mehrere Produktionen angegeben, die alternativ verwendet werden können.

4.1.2. Logikelemente für VLSI

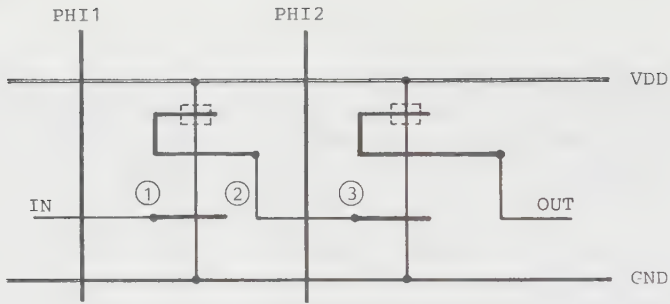
Die bisherigen Betrachtungen enthalten nur wenige, für die Höchstintegration spezifische Details. Bei der Realisierung von Schaltungen als VLSI-Schaltkreis ist es jedoch wesentlich, die spezifischen Eigenschaften hochintegrierter Schaltungen bereits im Entwurf adäquat darzustellen.

Transfergatter

Ein wesentliches Element von NMOS-Schaltungen sind Schalterfunktionen einzelner Transistoren, die eine sehr effiziente Realisierung logischer Funktionen erlauben. Diese sogenannten Transfergatter finden insbesondere beim Schalten von Datenpfaden und bei der Implementierung quasi-statischer Register und Latch-Register häufig Anwendung. Dabei wird ein Anreicherungs transistor als Schalter in den Datenpfad eingesetzt, der diesen genau dann freigibt, wenn durch Anlegen einer positiven Spannung der Kanal des Transistors leitend wird (MEAD 80).

Das in Bild 4.4 dargestellte Schieberegister zeigt eine typische Anwendung der Transfergatter zur Steuerung des Signalverhaltens durch Taktleitungen (PH11, PH12).

Der wesentliche Unterschied zwischen Transfergatter und einem AND-Gatter liegt im elektrischen Verhalten. Unabhängig von den Eingängen liefert ein AND Gatter immer einen definierten Ausgangswert. Legt man eine dreiwertige Logik wie in ERES (GARDILL 77) zugrunde, ist der Ausgang genau dann undefiniert, wenn mindestens einer der Eingangswerte undefiniert ist. Beim Transfergatter ist jedoch ein anders geartetes Phänomen zu beobachten. Bei Anliegen einer logischen Null am Gate des Transfergatters ist der Kanal unabhängig vom Eingangswert gesperrt, der



```
terminal OUT = not( if PHI1 then (not if PHI1 then IN));
```

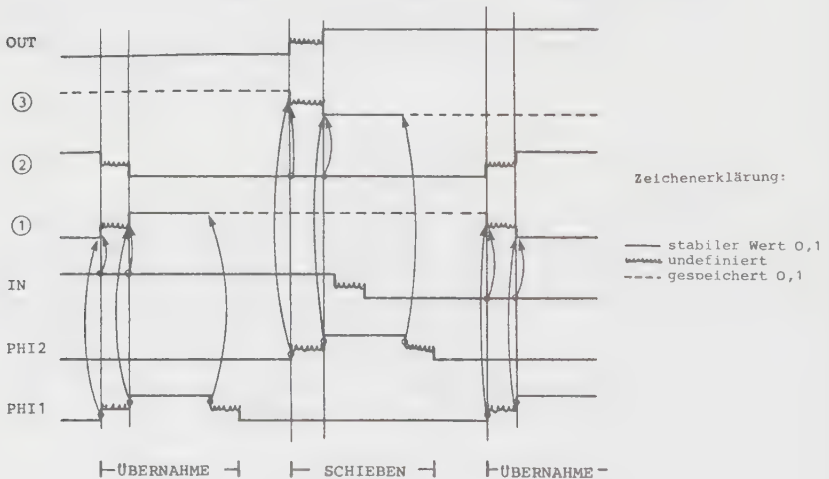


Bild 4.4: Schieberegister mit Transfergatter

Ausgang jedoch nicht undefiniert, sondern isoliert. Er speichert deshalb den zuletzt anliegenden Wert. Dieser Effekt beruht auf den parasitären Kapazitäten an den angesteuerten Gates der Eingangstransistoren anderer Gatter. Bei den heute verwendeten Technologien liegt die Zeitdauer der Speicherung durch die Leitungs- und Kanalkapazitäten dieser Transistoren um Größenordnungen über den Gatterlaufzeiten. Bei zukünftigen Technologien mit Kanallängen unter einem Mikron gelten die im folgenden

getroffenen Annahmen nur mit Einschränkungen (MEAD 80).

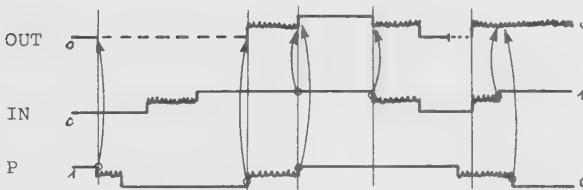
Der Speichereffekt isolierter Transistoren ist insbesondere im Hinblick auf verdrahtete Disjunktionen und die Realisierung von Registern von Bedeutung. Um eine realistische Beschreibung und Simulation zu ermöglichen wird für die weiteren Betrachtungen eine fünfwertige Logik mit folgenden Zuständen zugrunde gelegt:

0: logische Null, entspricht niederem Spannungspegel
 1: logische Eins, entspricht hohem Spannungspegel
 s0: gespeicherte 0
 s1: gespeicherte 1
 u: undefiniert, der logische Wert ist unbestimmt.

Die Wirkungsweise des Transferegatters unter Berücksichtigung des speichernden Verhaltens kann wie folgt charakterisiert werden:

terminal OUT = if C then IN;

dann gilt: $OUT(t) = IN$ falls $C=1$,
 $OUT(t) = OUT(t-1)$ falls $C=0$
 $OUT(t) = s0$ falls $C=u$ und $IN=0$ und
 $(OUT(t-1)=0$ oder $OUT(t-1)=s0)$,
 $OUT(t) = s1$ falls $C=u$ und $IN=1$ und
 $(OUT(t-1)=1$ oder $OUT(t-1)=s1)$,
 $OUT(t) = u$ falls $C=u$ und $OUT(t-1) \neq IN(t)$
 oder $C=s0$
 oder $C=s1$.



Beispiel 4.4: Signalverhalten am Transferegatter

Die letzte Bedingung bewirkt einen undefinierten Ausgangswert, falls ein Signalwechsel am Eingang während des Übergangszustandes des Passtransistors erfolgt oder das Transfergatter durch den Ausgang eines anderen Transfergatters angesteuert wird. Die Wirkungsweise wird an den in Beispiel 4.4 angegebenen Signalverläufen deutlich. Solange das Transfergatter leitend ist ($P=1$), wird der Eingangswert IN zum Ausgang OUT durchgeschaltet. Beim Übergang des Transfergatters von '1' nach 'u' behält der Ausgang seinen Wert in gespeicherter Form bei, wenn keine Änderung am Eingang erfolgt. Eine Änderung des Eingangs wirkt sich erst dann aus, wenn das Transfergatter wieder von '0' nach 'u' und dann nach '1' wechselt. Während der Übergangsphase ist der Wert des Ausgangs undefiniert, was eine 'worst case' Annahme darstellt. Tritt während einer Übergangsphase des Transfergatters vom leitenden in den sperrenden Zustand eine Änderung des Eingangs auf, bleibt der Ausgang bis zur nächsten leitenden Phase undefiniert.

Betrachtet man nun das Signalverhalten des in Bild 4.4 angegebenen Schieberegisters, wird deutlich, daß durch das speichernde Verhalten der beiden Transfergatter Änderungen am Eingang nur während der Übernahmephase Einfluß auf den Inhalt des Schieberegisters haben.

Verbindung von Terminals

Die Verbindung verschiedener Leitungen zu einem gemeinsamen Terminal ohne zwischengeschaltetes Gatter ist eine häufig anzutreffende Konstruktion, insbesondere wenn die beiden Quellen stets zu verschiedenen Phasen definierte Werte besitzen. Die andere Quelle ist dann meist durch ein Transfergatter gesperrt. Diese 'wired or' beziehungsweise 'wired and' Verknüpfungen sind technologieabhängig und nur mit einem exakten Modell hinreichend zu definieren. Für die hier zugrunde gelegte NMOS-Technologie und die zur Simulation verwendete fünfwertige Logik gilt folgende Verknüpfungstafel:

wired	0	1	s0	sl	u
0	0	u	0	0	u
1	u	1	1	1	u
s0	0	1	s0	u	u
sl	0	1	u	sl	u
u	u	u	u	u	u

Bemerkenswert ist dabei, daß gespeicherte Werte von logischen Signalen mit den Werten '0' oder '1' überdeckt werden. Das Ergebnis einer Verbindung ist nur dann undefiniert, wenn eines der Eingangssignale undefiniert ist oder die Signale unterschiedliche Werte 'gleicher Stärke' aufweisen.

Selektion von Terminals (Multiplexer)

Mit Hilfe der Definitionen von Transfergatter und Verdrahtung zweier Terminals läßt sich die Selektion eines Terminals durch einen Multiplexer als erweiterte Fallunterscheidung in Abhängigkeit von einem Selektor wie folgt beschreiben:

```
terminal OUT = if C then IN1 else IN0;
```

ist äquivalent zu:

```
terminal OUT = (if C then IN1) wired (if not C then IN0);
```

Die in den Datenpfad geschalteten Transfergatter geben jeweils nur eine der beiden Quellen frei. Die allgemeinere Form der Selektion kann aus der Grundform mit Hilfe einer rekursiven Definition abgeleitet werden. Dabei wird abhängig vom decodierten Wert der Selektionsbedingung aus einem Bus von Eingangssignalen genau eines selektiert und zum Ausgang durchgeschaltet. Wie in Rechnerentwurfssprachen üblich (BRENDÉL 83), erlauben wir bei der Terminal-Deklaration indizierte Bezeichner, um einen Bus gleichartiger Terminals zu beschreiben:

```

terminal OUT = case C(1..N) is
    0..00: IN(1),
    0..01: IN(2),
    ...
    1..11: IN(2**N);

```

ist äquivalent zu:

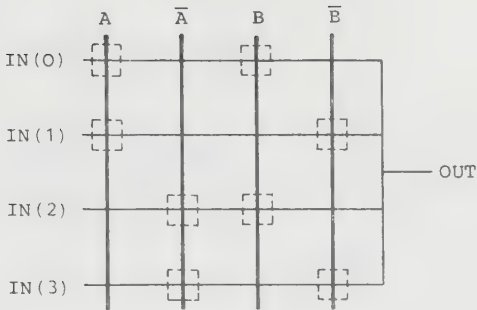
```

terminal OUT = if not C(1)
    then case C(2..N) is
        0..00: IN(1),
        ...
        1..11: IN(2**(N-1))
    else case C(2..N) is
        0..00: IN(2**(N-1)+1),
        ...
        1..11: IN(2**N);

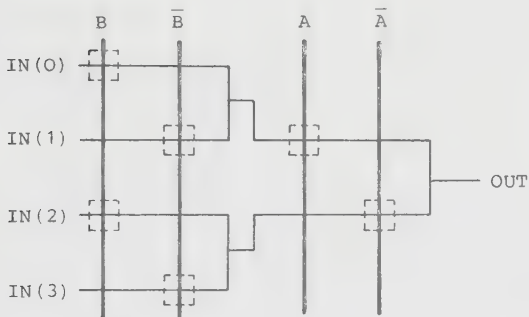
```

Die beiden äquivalenten Schaltungen eines Selektors für vier Eingänge ($N=2$) sind in Bild 4.5 dargestellt. Dabei entsteht durch die orthogonale Anordnung von Datenpfad und Selektorvariablen eine reguläre Struktur, die ein kompaktes Layout ergibt. An den Stellen, bei denen durch die Kreuzung der Selektorleitungen mit den Datenleitungen kein Transistor entstehen soll, wird durch Ionenimplantation eine unterbrechungsfreie Kreuzung geschaffen. Eine alternative Lösung unter Verwendung von Aluminium für die Selektorleitungen erlaubt schnellere Schaltzeiten, benötigt jedoch mehr Fläche (MAVOR 82).

Die Produktionen der Grammatik in Bild 4.2 zur Erzeugung der internen Struktur im Synthesystem sind für das Transfergatter und die Selektion zu erweitern. Bild 4.6 zeigt beide Produktionen, welche die Sprach-elemente der Entwurfssprache durch elementare Operatoren synthetisieren. Diese sind jedoch nicht technologieabhängig, da Schalterfunktionen neben der hier im Vordergrund stehenden NMOS-Technik auch in anderen Technologien realisiert werden können. Die Struktur des Kontrollgraphen ist entsprechend um die beiden Produktionen zu erweitern. Die Einschränkungen hängen dabei jedoch von der Implementierung des Synthesystems ab. Eine sinnvolle Restriktion ist, die Selektion nur als übergeordnete Struktur zur normalen Verknüpfungslogik zuzulassen.



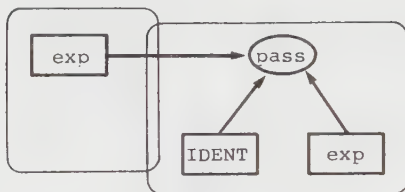
a) Selektion von vier Terminals



b) Äquivalente Form in zwei Stufen

Bild 4.5: Strichdiagramm Selektorschaltung

L8: Passtransistor



L9: Selektion

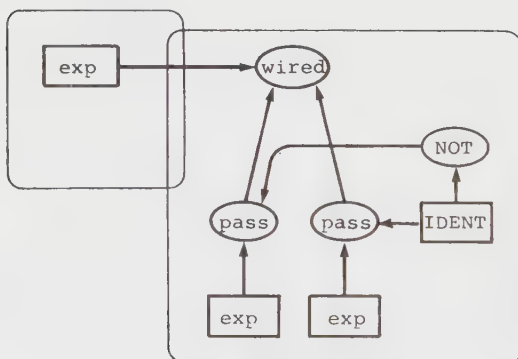


Bild 4.6: Ergänzung der Grammatik in Bild 4.2

4.2. REGISTER UND SPEICHER

Die Beschreibung von Registern und ihre Verwendung im Entwurf hängen wesentlich von Voraussetzungen ab, die bezüglich der Realisierung gemacht werden. Konventionelle TTL-Schaltungen enthalten meist flankengesteuerte Register eines bestimmten Typs, für deren Beschaltung es ausreicht, den Zeitpunkt der Übernahme der neuen Registerinhalte zu kennen. Diese Realisierung der Register in statischer Logik bewirkt zusätzlich, daß Registerinhalte unabhängig von der Taktfrequenz erhalten bleiben. Da die Register dabei durch zwei Stufen von Flip-Flops realisiert werden, ist eine größere Anzahl von Gattern notwendig, die zu einem größeren Flächenbedarf und geringeren Schaltgeschwindigkeiten führen (LEWIN 83). In hochintegrierten Schaltkreisen werden deshalb meist quasistatische Register verwendet, bei denen der Speichereffekt an den Gates der Eingangstristoren der angesteuerten Inverter zur Informationsspeicherung verwendet wird. Diese temporäre Speicherung setzt jedoch für den Betrieb der Schaltung eine gewisse minimale Grenzfrequenz für den Takt voraus (MAVOR 82).

Die ERES zugrunde liegende Modellvorstellung von Registern geht davon aus, daß die Übernahme neuer Information in ein Register zu einem idealisierten Zeitpunkt stattfindet, zu dem gleichzeitig der alte Registerinhalt noch gültig ist. Nach diesem Zeitpunkt wird das Register für die Dauer der spezifizierten Verzögerungszeit ungültig. Die Ansteuerung des Takteingangs der Register ist nur mit Takten, nicht jedoch mit logischen Signalen möglich. Dieses Registermodell ist für typische VLSI-Schaltkreise jedoch nicht mehr gültig und wird deshalb im folgenden wesentlich erweitert.

4.2.1. Register und Latch

Eine häufige Form der Realisierung von Registern in hochintegrierten Schaltungen sind quasistatische Register, die nur ein Drittel der Fläche traditioneller statischer Register benötigen. Dabei wird eine zweiphasige Arbeitsweise angewendet. In der ersten Phase erfolgt die Übernahme der neuen Information, während die Speicherung und das Durchschalten der Information des Registers in einer zweiten Phase vorgenommen wird. Bild 4.7 zeigt eine mögliche Realisierung eines

solchen quasistatischen 'master-slave' Registers in NMOS-Technologie.

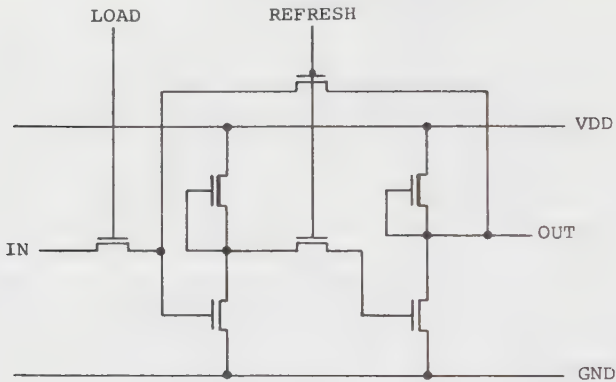


Bild 4.7: Register

Die Bezeichnung quasistatisch rührt daher, daß nur dann eine statische Speicherung der Information gewährleistet ist, wenn der Takt während der Wiederauffrischungsphase angehalten wird ($\text{REFRESH}=1$). Wesentlich für die Beschreibung in der Entwurfssprache ist der Effekt, daß während der Übernahme neuer Informationen die bisher gespeicherte am Ausgang stabil bleibt, bis die Speicherungs- und Wiederauffrischungsphase beginnt. Nach einer gewissen Verzögerung ist dann die neue Information am Registerausgang verfügbar. Da die Realisierung der Taktsteuerung durch Transferelemente vorgenommen wird, handelt es sich bei den Takten nicht um idealisierte Impulse, sondern um Signale mit einer endlichen Dauer (vergleiche Kapitel 5).

Prinzipiell unterschiedlich funktionieren die auch aus der TTL-Technik bekannten Latch-Register, bei denen bereits während der Übernahme der Information diese mit einer gewissen Verzögerung am Ausgang erscheint. Die Phase der Wiederauffrischung gespeicherter Information wird dabei ebenfalls durch einen zweiten Takt gesteuert, so daß ebenfalls ein quasistatisches Verhalten erreicht wird. Bild 4.8 zeigt eine typische Realisierung eines Latch.

Die Art des verwendeten Registers bestimmt die globale Struktur des

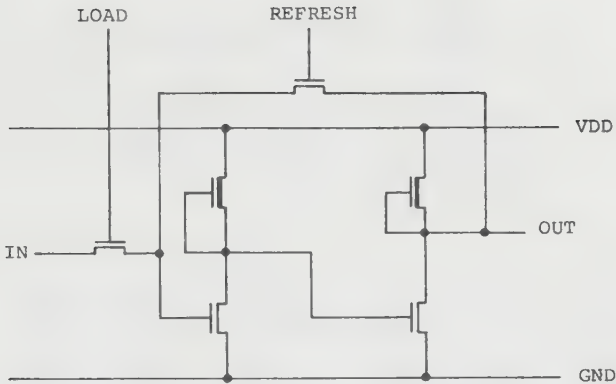
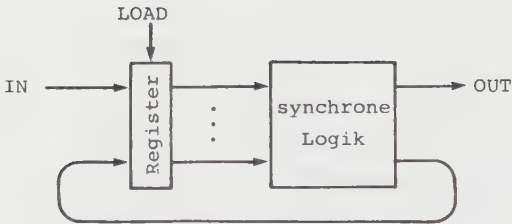


Bild 4.8: Latch

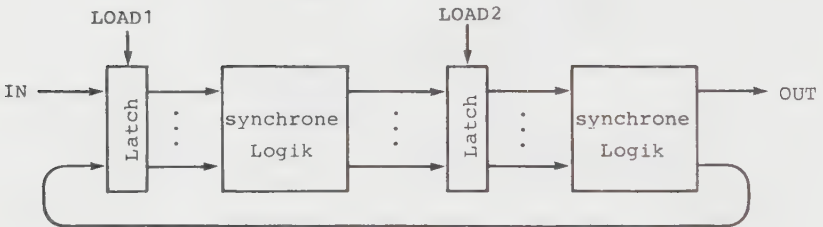
gesamten Entwurfs. Bei Registern gilt das in konventionellen Rechnerentwurfssprachen übliche Modell des zyklischen Austausches der Informationsinhalte zwischen synchroner Logik mit zwischengeschalteten Registern gleicher Taktansteuerung (Bild 4.9 a). Die Verwendung von Latch-Registern setzt eine Struktur mit Registern alternierender Taktansteuerung voraus (Bild 4.9 b), um asynchrone Laufzeitbedingungen auszuschießen.

Da die Beschreibung eines Entwurfs keine Voraussetzungen über die Art und die spezielle Struktur der verwendeten Register enthalten soll, ist es notwendig in den Register-Deklarationen zwischen Register und Latch zu unterscheiden und deren tatsächliche Struktur durch eine Spezifikation anzugeben. Wesentlich für die Beschreibung der Registertransfers ist dabei die Wahl der Takte für die Übernahme und die Speicherung der Information. Diese werden bei der Deklaration der Registerstruktur deshalb als formale Parameter angegeben, für die, bei der Beschreibung der Registertransfers, dann die entsprechenden Steuertakte eingesetzt werden (siehe Kapitel 6).

Die Syntax der Registerdekларation ist in Bild 4.10 dargestellt. Als Beispiel werden im folgenden die oben dargestellten Registerrealisierungen beschrieben. Die metasprachliche Variable 'source' bezeichnet



a) Register mit einem Takt



b) Latch-Register mit zwei Taktphasen

Bild 4.9: Schematische Darstellung synchroner Schaltwerke

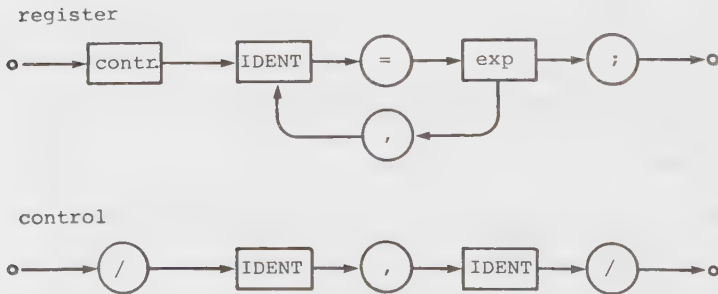


Bild 4.10: Syntax der Registerdeklaration

dabei den Inhalt des Terminals oder Registers mit dem das deklarierte Register bei Ausführung eines Registertransfers geladen wird.

Deklaration für das in Bild 4.7 dargestellte Register:

```
register /LOAD,REFRESH/
RS = not ( if REFRESH then
          not ( (if LOAD then source) wired (if REFRESH then RS) ));
```

Deklaration für das in Bild 4.8 dargestellte Latch:

```
latch /LOAD,REFRESH/
RD = not not ( (if LOAD then source) wired (if REFRESH then RD) );
```

Die Graphgrammatik der Registerdeklaration besteht aus einer Teilmenge der für die Terminaldeklaration verwendeten Produktionen. Zur Definition der Registerdeklaration ist deshalb lediglich ein zusätzlicher Kontrollgraph notwendig.

4.2.2. Adressierbare Speicher

Adressierbare Speicher weichen in ihrer Funktionsweise von den hier betrachteten synchronen Systemen wesentlich ab. Bei einem Schreibvorgang in den Speicher müssen Daten und Adressen zusammen mit einem Schreibsignal solange am Speicher anliegen, bis dieser den Schreibvorgang beendet hat. Ein Lesen des Speichers ist nur möglich, wenn die Adressen schon vorher am Speicher anlagen und die Verzögerungszeit des Speichers abgelaufen ist. Im Gegensatz zu Registern können jedoch verschiedene Teile des Speichers nicht zu einem Zeitpunkt gelesen oder beschrieben werden, da diese Vorgänge voneinander nicht unabhängig sind.

Arbeitet der Speicher zusätzlich mit Quittungssignalen, sind Mechanismen zur Synchronisation dieser Signale in der Entwurfssprache notwendig. Die sich bei einer Erweiterung von ERES um asynchrone Schnittstellen ergebende Problematik und Lösungsvorschläge enthält (GULDEN 83), so daß auf eine Einbeziehung dieser Problematik in der vorliegenden Arbeit verzichtet wird.

4.3. REGULÄRE STRUKTUREN

Wesentliches Merkmal hochintegrierter Schaltkreise ist die Verwendung regulärer Strukturen. Die Regularität eines Schaltkreises wird durch die mehrfache Verwendung der gleichen Zelle erhöht. Diese mehrfache Verwendung weniger Basiszellen ermöglicht den Entwurf komplexer Schaltungen mit relativ geringem Aufwand.

$$\text{Regularitätsfaktor} = \frac{\text{Gesamtzahl aller Transistoren}}{\text{Anzahl individuell entworfener Transistoren}}$$

Der Regularitätsfaktor ist ein häufig verwendetes Maß für die Beurteilung des Entwurfsaufwandes. Der in Kapitel 2 behandelte SORT-Baustein enthält 58 individuell entworfene Transistoren bei einer Gesamtzahl von 1498 Transistoren ohne periphere Treiber- und Schutztransistoren. Dies ergibt einen Regularitätsfaktor von 25,8. Bei einer Realisierung mit einer höheren Anzahl von Sortierelementen sind keine neuen Zellen zu entwerfen, da lediglich die vorhandene Struktur mehrmals wiederholt wird. Der Regularitätsfaktor für einen größeren SORT-Baustein würde deshalb wesentlich höher liegen.

Zur Beschreibung der Wiederholungen gleichartiger Zellen und der Spezifikation regulärer Implementierungen für umfangreiche logische Gleichungssysteme werden im folgenden verschiedene Kontrollstrukturen eingeführt.

4.3.1. Wortstruktur und Busse

Die häufigste Anwendung gleichartiger Zellen ist die Beschreibung der Wortstruktur von Registern oder einer Vielzahl gleichartiger Terminals, die zu einem Bus zusammengefaßt werden. In den meisten Rechner-Entwurfsprachen wird dies durch Angabe einer Indizierung bei der Deklaration des entsprechenden Elements beschrieben. Der Bezeichner eines Terminals oder Registers besteht deshalb aus einem eindeutigen Namen und dem, in Klammern stehenden, Indexbereich. Dieser gibt durch die Anzahl der zulässigen Indizes die Wortbreite des deklarierten Objekts an. Für jeden

Index werden Duplikate des deklarierten Objekts angelegt, deren einzelne Ausgänge bei der Verwendung in anderen Ausdrücken durch Angabe des Index spezifiziert werden. Wird bei der Verwendung eines Objekts nur ein Teil des Indexbereiches verwendet, so werden nur die Ausgänge der den Indizes entsprechenden Zellen als Informationsquelle herangezogen (vergleiche ERES, GARDILL 77).

4.3.2. Replikationsoperator

Die Indizierung ermöglicht die Wiederholung gleicher Zellen zwischen denen keine Querbezüge bestehen. Eine allgemeinere Form der Vervielfachung ist der bereits bei der Beschreibung des SORT-Bausteins verwendete Replikationsoperator ('forall .. is .. endall'). Dabei wird der Indexbereich einer Wiederholung explizit deklariert, wodurch innerhalb der Wiederholung Querbezüge zu anderen Indizes möglich sind. Das in Bild 4.11 dargestellte Beispiel einer Schaltung zur Bildung des Zweierkomplements hat bei Verwendung des Replikationsoperators folgende Spezifikation:

```

register R(1..16)    = ... ;
terminal COMPL      = ... ;

forall I = (15..1) is
    terminal    CARRY(I) = COMPL and ( R(I) or CARRY(I+1) ),
                S(I)     = R(I) xor CARRY(I+1),
                CARRY(16) = COMPL and R(16),
                S(16)    = R(16);
endall;
```

4.3.3. Reguläre Logikfelder

Die Realisierung logischer Gleichungen in Form von regulären, logischen Feldern (PLA: programmable logic array) ist eine Standardmethode beim Entwurf hochintegrierter Schaltkreise. Neben der hohen Regularität des Entwurfs und der leichten Änderbarkeit werden ein kompaktes Layout und hohe Schaltgeschwindigkeiten erzielt. Programmierbar ist ein solches Logikfeld nur im eingeschränkten Sinn, wie etwa ein ROM ('read only memory'), dessen Inhalt nur während der Entwurfsphase modifizierbar ist. Wurde die Transistorbelegung des Logikfeldes zur Fertigung endgültig

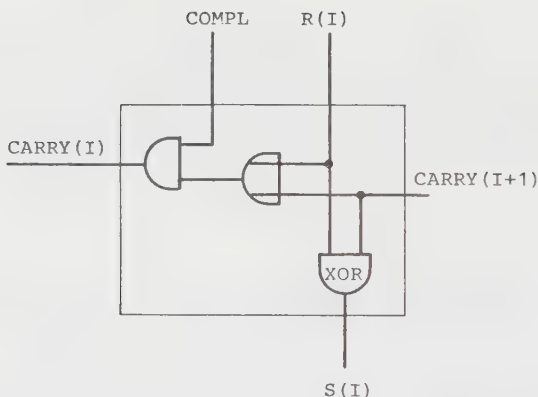


Bild 4.11: Schaltbild Komplementbildung

festgelegt, ist keine nachträgliche Änderung mehr möglich (Ausnahme ist das FPLA: 'field programmable logic Array').

Ausgangspunkt der Spezifikation eines PLA's ist eine disjunktive kanonische Normalform der logischen Gleichungen, deren Produktterme in einer AND-Stufe des PLA's zusammengefaßt werden. Eine zusätzliche OR-Stufe verknüpft dann die für jeden Ausgang relevanten Produktterme disjunktiv. Die Grundstruktur eines PLA und die Äquivalenztransformation der disjunktiven Normalform zur NOR-Form sind in Bild 4.12 dargestellt.

Die Ein- und Ausgänge des PLA's enthalten Inverterstufen, die mit geringen Modifikationen zu Registern erweitert werden können. Beschreibt man die Zustandsüberföhrungsfunktion eines deterministischen Automaten durch logische Gleichungen, die den aktuellen Zustand als Eingangsvariablen und den Folgezustand als Ausgangsvariablen enthalten, kann ein deterministischer Automat ebenfalls als PLA realisiert werden. Ein Synthesystem, das ausschließlich auf der Konstruktion von PLA's für die Realisierung eines gesamten Entwurfs beruht, beschreibt (AYRES 83). Eine eingehende Erläuterung der Funktionsweise eines PLA's enthalten (MEAD 80, MAVOR 82). In diesem Abschnitt wird lediglich auf die Realisierung logischer Gleichungen durch PLA's eingegangen, während in Abschnitt 6.2

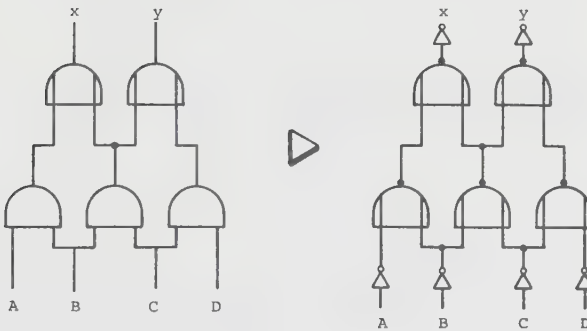
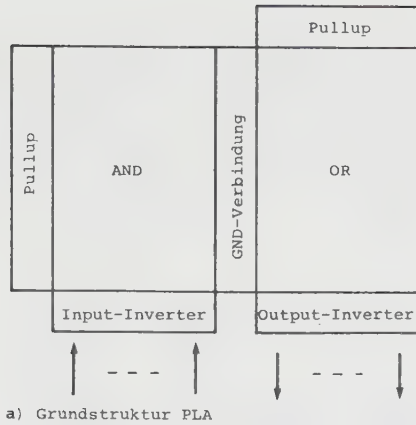


Bild 4.12: Grundstruktur PLA

die Realisierung von Kontrollstrukturen für Automaten behandelt wird. Für die folgenden Erläuterungen wird die Schaltung der lokalen Steuerlogik des SORT-Bausteines aus Abschnitt 2.3.2 als Beispiel herangezogen.

Da ein PLA mehrere logische Gleichungen mit einer gemeinsamen Menge von

Eingangsvariablen in einer geschlossenen regulären Struktur realisiert, muß in der Entwurfssprache eine Deklaration eingeführt werden, die diese Gleichungen syntaktisch zusammenfaßt. Darüber hinaus sind zur Spezifikation nur Ausdrücke in disjunktiver Normalform zulässig, um die Abbildung der Beschreibung auf die Realisierung transparent zu halten. Dies ermöglicht bereits aufgrund der funktionalen Beschreibung eine Abschätzung der im PLA enthaltenen Produktterme und OR-Ebenen und somit eine Ermittlung des zu erwartenden Flächenbedarfs. Das Syntaxdiagramm für die Spezifikation eines Logikfeldes zeigt Bild 4.13.

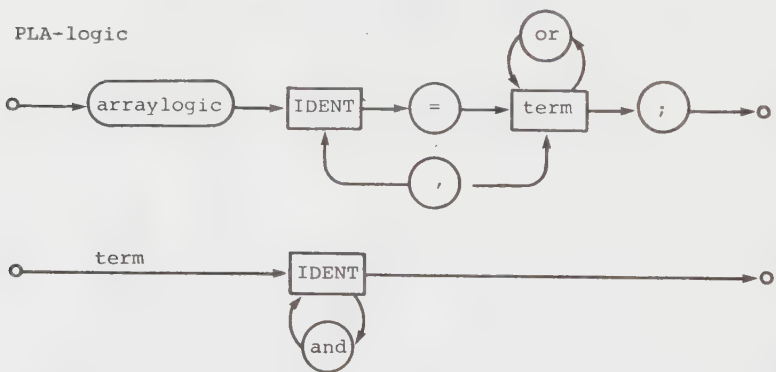


Bild 4.13: Syntaxdiagramm Logikfeld.

Die Deklaration der lokalen Steuerlogik aus dem SORT-Baustein als Logikfeld ist eine direkte Umformung der NAND/NOR-Gleichungen aus Abschnitt 2.3.2.:

```

arraylogic  UPIN  = (PUSHN and not POPN) or (PUSHN and SWAP),
              DOWNIN = (POPN and not PUSHN) or (POPN and SWAP);

```

Da bei der Realisierung eines PLA's zum Treiben der langen Leitungen Inverter vorgeschaltet werden, stehen alle Eingangsvariablen sowohl in negierter als auch in nichtnegierter Form zur Verfügung. Damit sind bei negierten Eingängen keine zusätzlichen Inverter notwendig.

Das obige Beispiel enthält insgesamt vier Produktterme, von denen

jeweils zwei disjunktiv verknüpft werden. Bei der Beschreibung größerer Gleichungssysteme sind meist mehrere Produktterme in verschiedenen Disjunktionen verwendet, so daß eine eventuelle Optimierung nicht wie bei klassischen Optimierungsverfahren auf die Anzahl der Operationen, sondern vielmehr auf eine Kompaktifizierung der topologischen Struktur abzielt (SEGOVIA 82).

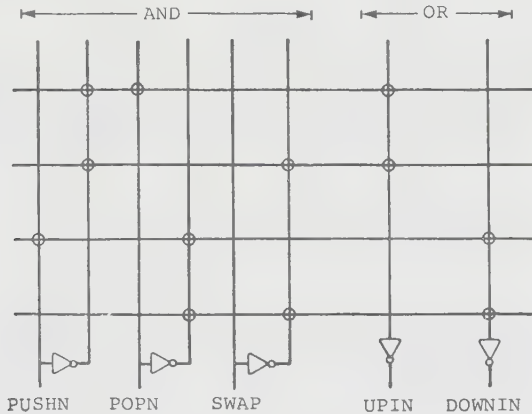
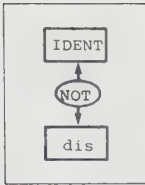


Bild 4.14: Symbolisches PLA der lokalen Steuerlogik

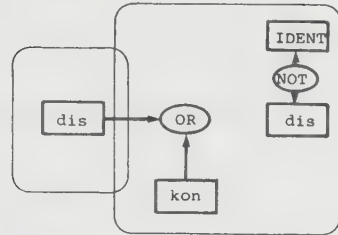
Die in Bild 4.14 dargestellte symbolische Struktur des PLA's zum Beispiel der lokalen Steuerlogik enthält für jeden Produktterm eine Zeile, die bei den entsprechenden Eingangsvariablen in der AND-Stufe einen Transistor aufweist. Die Zusammenfassung der Disjunktionen in der OR-Stufe ordnet jeder Ausgangsvariablen die Produktterme zu, die in der entsprechenden Gleichung auftreten.

Die Transformation der PLA-Deklaration in einen Entwurfsgraphen (Bild 4.15) ergibt vier Ebenen von Knoten, wobei die unterste Ebene von den Eingangsvariablen, die oberste Ebene von den Ausgangsvariablen und die dazwischenliegenden durch die AND und OR-Verknüpfungen gebildet werden. Die entsprechenden Produktionen zum Aufbau dieses Graphen aus der funktionalen Beschreibung zeigt Bild 4.16.

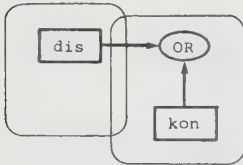
Startgraph:



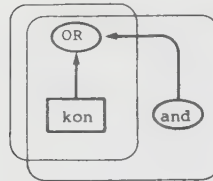
LA1: Zusätzliche Ausgänge



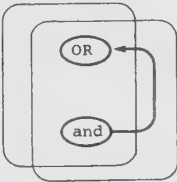
LA2: Abschluß



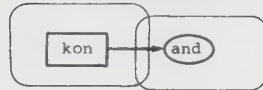
LA3: Zusätzliche Terme eines Ausgangs



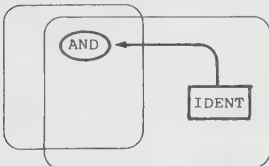
LA4: Vorhandene Terme



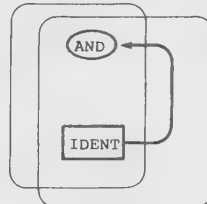
LA5: Abschluß Terme



LA6: Neuer, nichtnegierter Eingang



LA8: Vorhandener, nichtnegierter Eingang

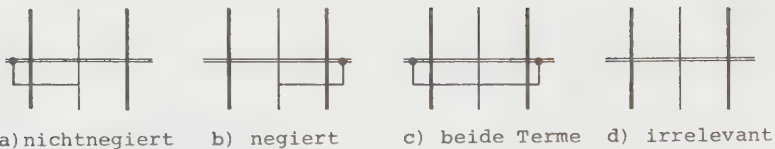


LA7: Neuer negierter Eingang analog LA5

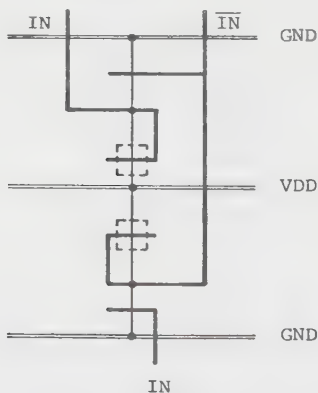
LA9: Vorhandener, negierter Eingang analog LA8.

Bild 4.16: Grammatik für die Spezifikation von Logikfeldern

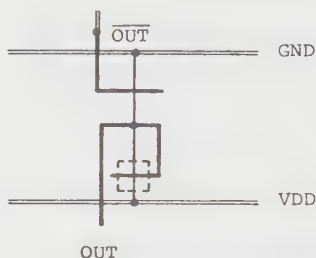
1) Grundelement für AND/OR-Ebene



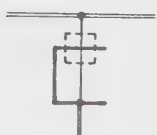
2) Eingangsstufe



3) Ausgangsstufe



4) Pull-up Element



5) GND, Term-Übergang



6) GND-Versorgung



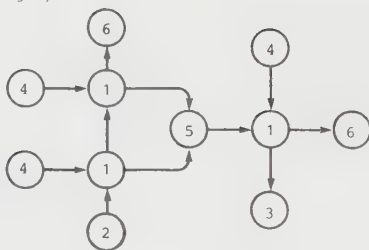
Bild 4.17: Strichdiagramme der PLA-Zellen

Der Kern eines PLA mit den Transistoren in der AND- und der OR-Ebene besteht prinzipiell aus einer Grundzelle, die entsprechend dem Auftreten der Eingangsvariablen (negiert, nichtnegiert oder irrelevant) und den, zur Bildung der Disjunktionen verwendeten Termen, durch Transistoren

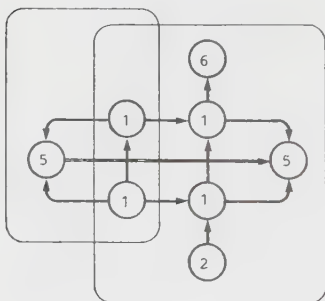
ergänzt wird. Die restlichen fünf Zellen sind die Treiberstufen für Ein- und Ausgänge mit integrierten Invertern, die Lasttransistoren zum Treiben der Leitungen sowie Zellen zur Masseverteilung an der Peripherie des PLA's. Die Strichdiagramme für das Zell-Layout zeigt Bild 4.17. Wesentlich bei der Realisierung des Layouts ist, daß die Dimensionierung der Zellen so bemessen wird, daß das gesamte Layout ohne zusätzliche Verdrahtung durch einfaches Aneinanderfügen der Zellen erfolgen kann. Zur Vereinfachung der Produktionen für die Beschreibung der Zellgrammatik wird zwischen den einzelnen Varianten der PLA-Grundzelle mit oder ohne Transistoren nicht unterschieden. Diese sind jeweils alternativ zu verwenden.

Da die Grammatik zur Synthese eines PLA (Bild 4.18.a und 4.18.b) beliebige reguläre PLA's erzeugen soll, ist es, im Gegensatz zur Grammatik der Entwurfssprache, nicht notwendig, beim Einsetzen der Transistorzellen die Generierung aufgrund der verwendeten logischen Gleichungen zu steuern. Die Produktionen der PLA-Grammatik und der zugehörige Kontrollgraph erzwingen lediglich die Einhaltung der regulären Struktur und die Erzeugung der entsprechenden peripheren Zellen. Die notwendige Symmetrie der Zellen, die zur Synthese der AND und der OR-Ebene verwendet werden, erfordert jedoch die paarweise Anordnung der Term-Zeilen. Dadurch werden immer Strukturen erzeugt, die eine gerade Anzahl von Termen aufnehmen können. Dies muß beim Entwurf der PLA-Deklaration berücksichtigt werden, da andernfalls bei der Synthese Fehlermeldungen auftreten könnten.

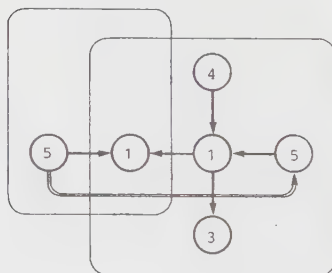
Startgraph



PLA1: Zusätzliche Eingänge



PLA2: Zusätzliche Ausgänge



PLA3: Terme, Mitte

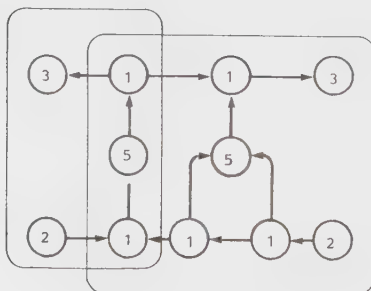
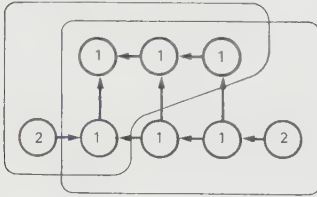
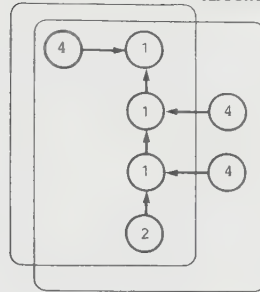


Bild 4.18.a: Startgraph und PLA Produktionen 1 bis 3

PLA4: Terme, AND-Stufe

PLA5: Terme, AND-Stufe
Abschluß

PLA6: Terme, OR-Stufe

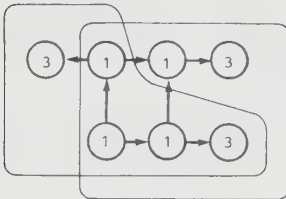
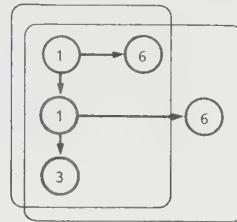
PLA7: Terme, OR-Stufe
Abschluß

Bild 4.18.b: PLA Produktionen 4 bis 7

Der Kontrollgraph zur Grammatik der Synthese eines PLA's aus den obigen Zellen ist in Bild 4.19 dargestellt. Die Bezeichnung der Knoten bezieht sich auf die entsprechenden Nummern der Strichdiagramme in Bild 4.17. Für die Grundelemente des PLA's (Knoten 1) kann jede der vier Varianten a-d eingesetzt werden. Um die Grammatik der Zellen klein zu halten wurde auf eine explizite Darstellung der Varianten verzichtet.

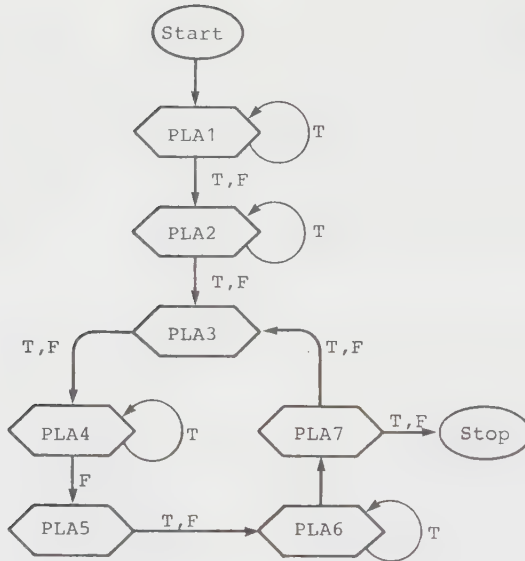


Bild 4.19: PLA Kontrollgraph

Die Struktur des PLA's für die lokale Steuerlogik ist in Bild 4.20 als Graph mit den entsprechenden Zellen dargestellt. Um eine bessere Zuordnung zur Realisierung zu ermöglichen, sind die tatsächlich verwendeten Varianten der PLA-Grundelemente eingetragen.

Die Verwendung eines PLA's in hochintegrierten Schaltkreisen ist besonders bei komplexen Gleichungen oder bei einer Vielzahl von Ausgängen vorteilhaft, die durch eine vergleichsweise geringe Anzahl von Eingängen bestimmt werden. Die regelmäßige Struktur ermöglicht eine kompakte Realisierung, deren Platzbedarf bei geringen Änderungen der Logik nahezu gleich bleibt. Das obige Beispiel der lokalen Steuerlogik des SORT-Bausteines ist nur zur Demonstration der Vorgehensweise bei der Synthese gedacht, da die Realisierung als PLA wegen des großen Platzbedarfs der peripheren Zellen, einer Realisierung durch freie Logik unterlegen ist. Das in Bild 4.21 als Strichdiagramm dargestellte PLA der lokalen

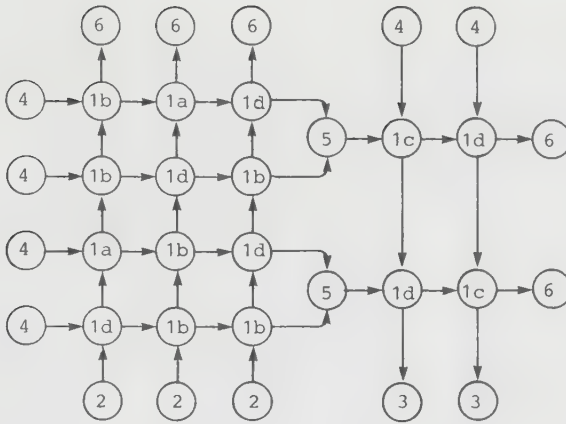


Bild 4.20: Struktur der lokalen Steuerlogik

Steuerlogik zeigt dies beim Vergleich mit dem in Abschnitt 2.3.2 gezeigten Entwurf deutlich.

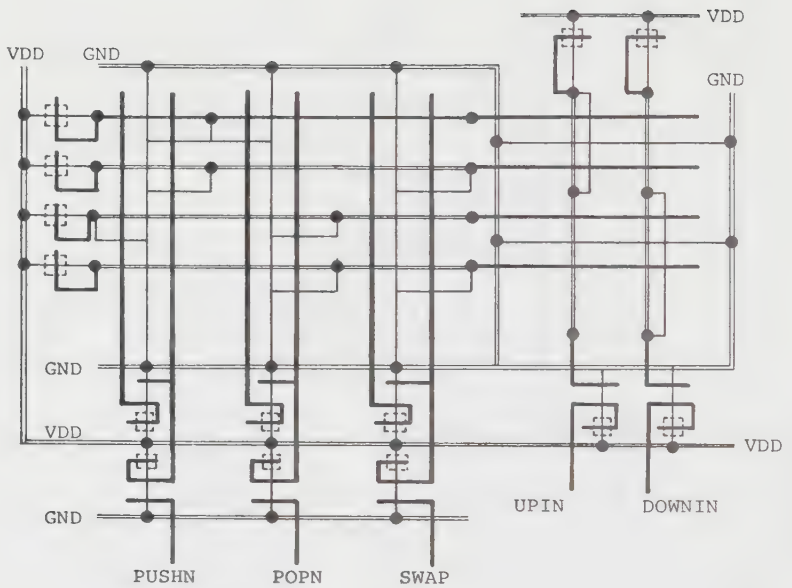


Bild 4.21 Strichdiagramm PLA lokale Steuerlogik

The successful design of large scale integrated systems requires careful management not only of the two-dimensional silicon area but also of the operation of the system in the time dimension.

C.L.Seitz

5. DEFINITION UND SIMULATION DES ZEITMODELLS

Das Zeitmodell für die Beschreibung und Simulation hochintegrierter Schaltkreise geht von einer prinzipiell synchronen Arbeitsweise des Entwurfs aus, wie sie bei den heutigen Schaltkreisen weit verbreitet ist. Dabei ist ein Entwurf nur dann synchron bezüglich eines vorgegebenen Zeittaktes, wenn innerhalb der Verknüpfung keine Rückführungen von Signalen und damit asynchrone Rückkopplungen auftreten. Bei der Deklaration der logischen Verknüpfungen als 'terminal' oder 'logic array' wurde dies bereits ausgeschlossen. Dies bedingt, daß jede Rückkopplung eines Signales durch ein zwischengeschaltetes Register aufgelöst werden muß. Diese Einschränkung ist nicht nur in Hinblick auf eine effiziente Implementierung des Simulators sinnvoll, sondern zur Konstruktion testfreundlicher Schaltungen notwendig (siehe Abschnitt 7.2).

5.1. SPEZIFIKATION VON TAKTEN

Das ERES Zeitmodell (KLAR 74) abstrahiert reale Taktsignale als idealisierte Impulse, ohne zeitliche Ausdehnung. Wie bereits in Kapitel 4 erläutert setzt dies ein Registermodell voraus, bei dem in einem Entwurf bezüglich der Taktansteuerung gleichartige Register verwendet werden, die zu definierten Zeitpunkten neue Werte von den Eingängen übernehmen, ohne daß die Ausgänge dadurch nahezu gleichzeitig beeinflußt werden. Dadurch kann ein Register zu einem Zeitpunkt geladen und gelesen werden. Bei den für hochintegrierte Schaltkreise typischen Latch-Registern trifft dies jedoch nicht zu. ERES wird deshalb so erweitert, daß Takte ('clock') in der Modellbildung eine endliche Ausdehnung aufweisen, die dem realen Takt entspricht.

5.1.1. Takte und Mehrphasenuhr

ERES ermöglicht eine präzise Beschreibung der Uhr und der Verzögerungszeiten vergleichbar mit CASSANDRE (ANCEAU 69) und CONLAN (PILOTY 83). Besonderes Merkmal von ERES ist jedoch die Erzeugung eines effizienten Simulators, der es gestattet, einen Entwurf samt der Mikroprogramme zu simulieren (BRENDDEL 82, BRENDDEL 83). Da die speichernden Elemente eines Entwurfs, die Register, sich nur zu definierten Taktzeitpunkten ändern können, reicht es aus, die Simulationszeit im Taktraster fortzuschalten, ohne für alle dazwischenliegenden Zeitpunkte die Werte der Register und Verknüpfungen zu berechnen. Vielmehr kann bei jedem Taktzeitpunkt die Gültigkeit der Quellen eines Registers aufgrund der, für die logischen Verknüpfungen angegebenen, Verzögerungszeiten berechnet werden.

Die Modellierung von Takten durch Impulse definiert exakt die Zeitpunkte, bei denen während der Simulation eine solche Berechnung erfolgen muß. Wesentliche Voraussetzung ist dabei jedoch, daß die durch die Takte spezifizierten Aktivierungsbedingungen für den Registertransfer stets definierte Zustände aufweisen. Um trotz der Erweiterung des Taktmodells eine effiziente Implementierung der Simulation zu ermöglichen, werden die Takte in CERES zwar mit endlicher Ausdehnung, jedoch ohne einen undefinierten Zustand modelliert (nur '0' und '1'). Bild 5.1 zeigt die Repräsentation eines Zweiphasentaktes durch Taktsignale mit idealisiertem Zustandsübergang, der keine undefinierten Zwischenwerte aufweist. Diese Modellvorstellung ist auch bei der Beschreibung hochintelligenter Schaltungen zulässig, da die gewöhnlich extern durch einen Quarz erzeugten Takte wesentlich steilere Flanken aufweisen, als dies für die in logischen Verknüpfungen entstehenden Signale der Fall ist.

5.1.2. Abgeleitete Taktsignale

In einem synchronen Entwurf ist nur die Verwendung einer Uhr mit eventuell mehreren Phasen zulässig (Mehrphasenuhr), da andernfalls nicht ausgeschlossen werden kann, daß eine Verschiebung der verschiedenen Taktraster auftritt. Zur Beschreibung von Aktivitäten, die zwar synchron zum Taktraster, jedoch nicht regelmäßig auftreten, ist es notwendig, Takte mit logischen Signalen zu verknüpfen, um eine Teilmenge der Uhrphasen zu definieren. Sie werden durch Verknüpfung von Takten der Mehrphasenuhr mit anderen Signalen spezifiziert. Diese abgeleiteten Taktsignale ('tick') sind im allgemeinen nicht periodisch, haben ansonsten

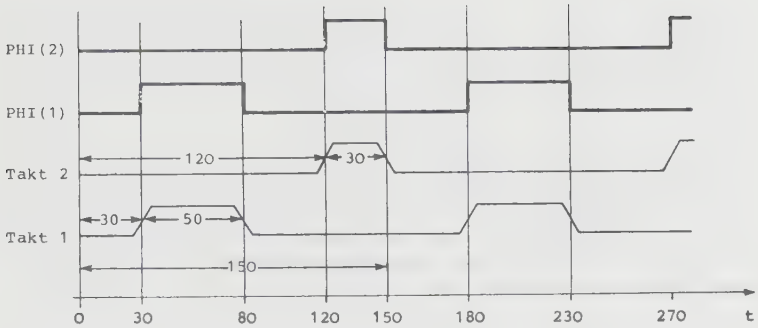


Bild 5.1: Modellierung realer Takte

jedoch die Eigenschaften idealisierter Takte, insbesondere sind sie stets definiert. Ein Beispiel für abgeleitete Taktsignale zeigt Bild 5.2. Der Takt T12 entsteht dabei durch eine disjunktive Verknüpfung der beiden Phasen $\text{PHI}1$ und $\text{PHI}2$, während T1SEL durch die Konjunktion der Phase $\text{PHI}1$ mit einem logischen Signal SELECT definiert ist.

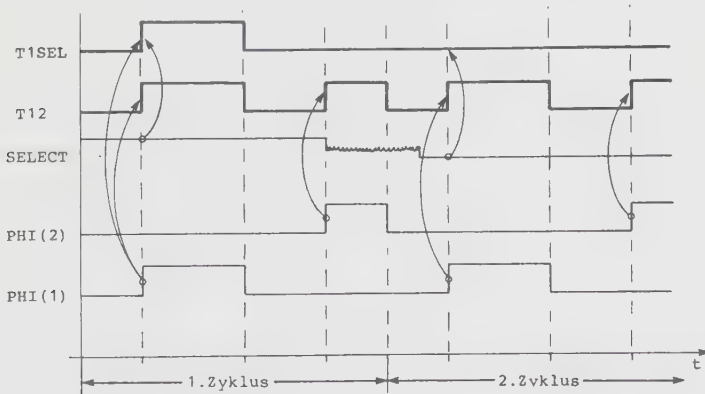


Bild 5.2: Abgeleitete Taktsignale

Neben der Einschränkung, daß abgeleitete Takte stets definierte Zustände aufweisen, sind für eine effiziente Simulation und einen strukturierten Entwurf weitere Eigenschaften notwendig, die sich auf die Syntax der 'tick'-Deklaration auswirken.

Da abgeleitete Takte wie Takte der Mehrphasenuhr zur Steuerung der Registertransfers verwendet werden sollen, müssen sie stets die Werte '0' oder '1' aufweisen. Die Ableitung kann deshalb nur durch eine Verknüpfung mit einem Takt der Mehrphasenuhr oder einem anderen, abgeleiteten Takt erfolgen. Die Signale, welche zur Ableitung herangezogen werden, müssen während der gesamten Taktphase einen definierten Wert aufweisen, um die Definiertheit des abgeleiteten Taktes sicherzustellen. Eine Ableitung durch Verwendung von Transfergattern (if...then...else') oder einer Selektorschaltung ('case...is...') ist nicht zulässig, da sonst zusätzlich die Werte 's0' und 's1' auftreten könnten. Diese sind jedoch nicht zur Ansteuerung eines weiteren Transfergatters geeignet.

Da die Einführung abgeleiteter Takte nur die Möglichkeit bieten soll bestimmte Takte der Mehrphasenuhr zusammenzufassen oder nichtperiodische Teilfolgen zu erzeugen, ist der abgeleitete Takt nicht mit einer Verzögerung behaftet. Andernfalls würden bei abgeleiteten Takten zusätzliche Zeitpunkte zwischen den Takten der Mehrphasenuhr entstehen, bei denen im Rahmen der Simulation eine Berechnung der Zustände erfolgen müßte.

Die Deklaration der Mehrphasenuhr erfolgt durch Angabe des Bezeichners und der Dimension (Anzahl der Takte). Der Signalverlauf des Taktes wird durch den zeitlichen Abstand zum Beginn eines Zyklus und die Dauer der Aktivierungsphase beschrieben. Die Dauer des Zyklus einer Mehrphasenuhr ergibt sich aus der Summe der beiden Werte der letzten Taktphase. Die Ableitung weiterer Takte wird in der 'tick'-Deklaration ohne Angabe zusätzlicher Zeitintervalle spezifiziert.

Die in Bild 5.1 und 5.2 dargestellten Takte können mit folgenden Deklarationen spezifiziert werden:

```
'clock' :30/50,120/30: PHI(1..2);
'tick'  T12      = not (PHI(1) nor PHI(2)),
        T1SEL    = not (PHI(1) nand SELECT) ;
```

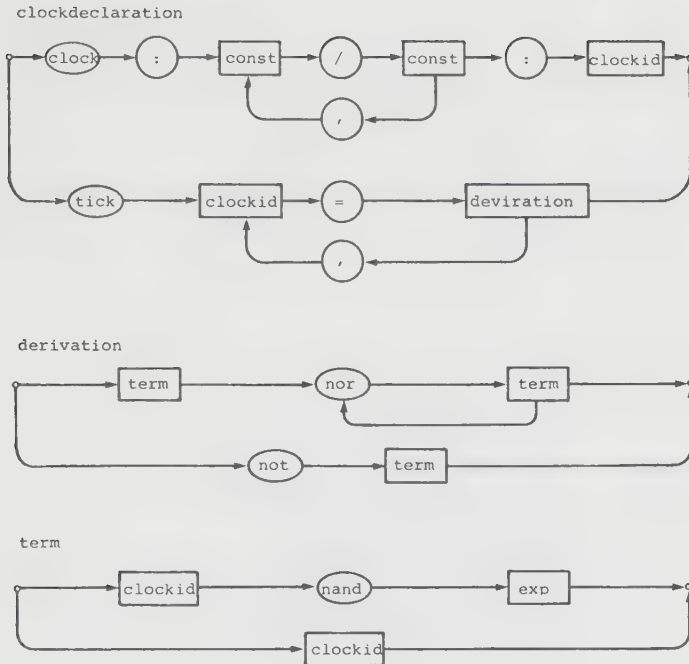


Bild 5.3: Syntaxdiagramm Taktdeklaration

5.2. Verzögerungszeiten

Eine funktionale Spezifikation hochintegrierter Schaltkreise auf der Register-Transferebene mit Hilfe einer Rechner-Entwurfssprache ist nur dann von hinreichender Präzision, wenn neben der strukturellen und funktionalen Beschreibung auch eine Darstellung des zeitlichen Verhaltens mit dem gleichen Entwurfshilfsmittel möglich ist. Die funktionale Beschreibung des Entwurfs ist nicht nur ein Dokumentationsmittel, sondern vielmehr Werkzeug zur Bewertung und Optimierung des Entwurfs auf

einer möglichst hohen Abstraktionsebene. Dabei ist es von geringem Interesse, welche Verzögerungszeiten an einzelnen Gattern des Entwurfs auftreten, da diese Angaben keinen Überblick über den zeitlichen Anteil der einzelnen Funktionsabläufe gestatten. Die in ERES angewandte Strategie, stets vollständigen logischen Verknüpfungen eine Gesamtverzögerungszeit zuzuordnen, hat sich auch bei der Anwendung auf die Beschreibung hochintegrierter Schaltkreise bewährt, da hier die einzelnen Signalwechsel so überlagert wirken, daß eine detaillierte Zuordnung der Gatterlaufzeiten zu großen Abweichungen gegenüber dem tatsächlichen Schaltverhalten führen würde. Hinzu kommt bei der hier im Mittelpunkt stehenden NMOS-Technologie ein asymmetrisches Verhalten der Signale, das beim Zustandswechsel von '0' nach '1' ein mehrfaches der Zeit benötigt wie der umgekehrte Übergang. Bei einer globaleren Zeitabschätzung werden diese Unterschiede subsummiert, da innerhalb eines Terminals beide Zustandsübergänge vorkommen.

Aufgrund technologischer Eigenschaften des Herstellungsprozesses treten im allgemeinen größere Schwankungen der Schaltzeiten einzelner Gatter auf, die durch eine entsprechende Abschätzung in der Deklaration kompensiert werden müssen, wenn die Ausbeute der Fertigung hinreichend groß sein soll. Neben diesen fertigungstechnischen Einflüssen sind jedoch auch Schwankungen der Schaltgeschwindigkeiten bei unterschiedlichen Betriebstemperaturen zu berücksichtigen. Unter Beachtung all dieser Einflußgrößen ist es nicht sinnvoll, zur Spezifikation der Verzögerungszeit exakt die Ergebnisse einer elektrischen Simulation der einzelnen Gatter zu verwenden. Eine Zeitabschätzung in der funktionalen Beschreibung, die von Laufzeiten ausgeht, die der Summe von Erwartungswert und dem dreifachen der Standardabweichung entsprechen (KLEINERT 83), läßt genügend Spielraum, um zu erwartende Schwankungen zu kompensieren, ohne zu Effizienzverlusten zu führen (MURRMANN 79). Diese statistischen Größen sind jedoch nur durch eine genaue Abstimmung der elektrischen Simulation mit dem verwendeten Prozess zu ermitteln.

5.2.1. Zeitverhalten von Register und Latch

Das zeitliche Verhalten von Register und Latch unterscheidet sich durch die Verzögerungszeit mit der nach dem Laden der Register der neue Wert am Ausgang beobachtet werden kann. Während beim Register der alte Inhalt während der Ladephase am Ausgang noch zur Verfügung steht, verhält sich

das Latch wie ein Terminal. Das Eingangssignal tritt mit einer gewissen Verzögerung auch am Ausgang auf und vollzieht auch eventuelle Signalwechsel nach. Erst mit Beendigung der Ladephase (Transferrgatter sperrt den Latch-Eingang) wird der zuletzt anliegende Wert bis zur nächsten Ladephase gespeichert. Gemeinsam ist beiden Registertypen, daß bei der Realisierung in einem hochintegrierten Schaltkreis eine zweiphasige Ansteuerung für Lade- und Wiederauffrischungsphase notwendig ist, da zur Speicherung der Information die Kapazitäten an den Transistoren ausgenutzt werden, die dynamisch aufgefrischt werden müssen (quasistatische Register).

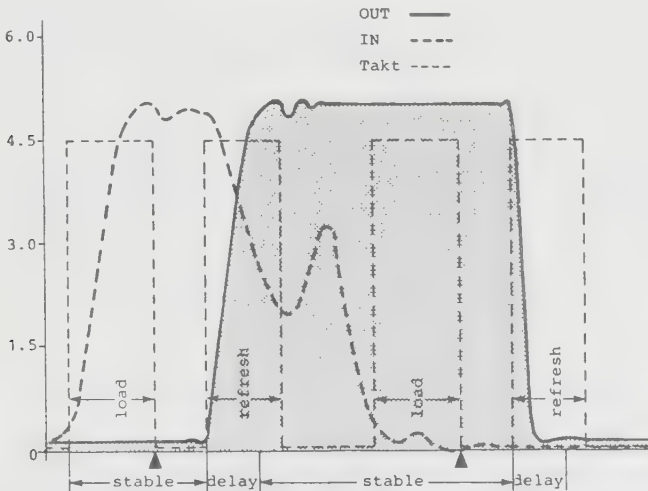


Bild 5.4: Zeitverhalten eines Registers

```

register :80: /LOAD,REFRESH/
      RS = not ( if REFRESH then
                not ( (if LOAD then source)
                    wired (if REFRESH then RS) ) );

```

Register sind zweistufig aufgebaut, indem die erste und die zweite Stufe durch ein Transferrgatter getrennt sind. Dadurch wird erreicht, daß der

Ausgang bis zur Wiederauffrischungsphase stabil bleibt und der alte Wert bei Übernahme eines neuen noch einige Zeit zur Verfügung steht (siehe Bild 4.7). Bei Einsetzen der Wiederauffrischungsphase (REFRESH) wird der neue Wert zum Ausgang durchgeschaltet und erscheint mit einer gewissen Verzögerung am Ausgang des Registers (siehe Bild 5.4). Vom Beginn der Wiederauffrischungsphase bis zu dem Zeitpunkt, an dem der neue Wert am Ausgang stabil ist, ist der Wert des Registers undefiniert. In Bild 5.4 ist deutlich erkennbar, daß der Übergang des Ausgangs von '0' nach '1' deutlich langsamer vonstatten geht als der umgekehrte Übergang im zweiten Zyklus. Zur Beschreibung des zeitlichen Verhaltens des Registers genügt es, die Verzögerungszeit vom Beginn der Deklaration anzugeben. Die Gesamtverzögerungszeit des Registers vom Laden bis zum Auftreten des neuen Wertes am Ausgang ergibt sich dann aus der Summe des zeitlichen Abstandes zwischen Lade- und Wiederauffrischungsphase sowie dem in der Deklaration angegebenen Verzögerungswert. Dies führt dazu, daß die tatsächliche Gesamtverzögerung vom Simulator in Abhängigkeit von den in den Aktivierungsbedingungen angegebenen Takten berechnet werden kann.



Bild 5.5: Zeitverhalten eines Latch

Im Gegensatz zum Register wird beim Latch der Eingangswert bereits

während der Ladephase (LOAD) mit einer gewissen Verzögerung zum Ausgang durchgeschaltet. Der Ausgang bleibt dann nach Beendigung der Ladephase und Ablauf der Verzögerungszeit bis zur nächsten stabil und zwar unabhängig vom weiteren Signalverlauf am Eingang (siehe Bild 5.5). Der Ausgang eines Latch folgt damit während der Ladephase dem Eingang mit der spezifizierten Verzögerungszeit und verhält sich bis zum Ende der Ladephase wie ein Terminal. Dies führt dazu, daß der alte Wert während der Ladephase praktisch sofort verloren geht und das Latch nicht wie ein Register gleichzeitig geladen und gelesen werden kann.

```
register :50: /LOAD,REFRESH/
      RD = not not ( (if LOAD then source) wired
                    (if REFRESH then RD) );
```

Eine 'worst case' Betrachtung des Signalverhaltens am Latch zeigt, daß der Ausgang während der Ladephase und der anschließenden Verzögerungszeit undefiniert ist. In der Simulation ist es deshalb nicht notwendig, alle Zeitpunkte diskret zu berechnen, sondern es genügt, die spezifizierte Verzögerungszeit ab dem Ende der Ladephase zu berechnen. Der Ausgang des Latch ist dann bis zur nächsten Ladephase stabil.

5.2.2. Terminals

Wie bereits in den vorangegangenen Bemerkungen zu diesem Abschnitt erläutert, wird bei der Deklaration von Terminals die Verzögerungszeit pauschaliert für die gesamte logische Verknüpfung angegeben. Die tatsächliche Verzögerungszeit hängt jedoch von den internen Zuständen der Gatter und von Anzahl und Wert der Eingangsvariablen ab. Ändert sich in einer umfangreichen Verknüpfung eine Eingangsvariable des letzten Gatters, ist die Verzögerungszeit im Verhältnis zum schlechtesten Fall, bei dem sich der geänderte Wert einer Eingangsvariablen durch alle Gatter der Verknüpfung fortpflanzt, gering. Eine funktionale Simulation mit einer beschränkten Anzahl von Testfällen liefert jedoch nur dann zuverlässige Ergebnisse, wenn auf diese Effekte verzichtet wird und stets die gesamte spezifizierte Verzögerungszeit zur Berechnung der Laufzeiten herangezogen wird.

Im Gegensatz dazu ist es wichtig, zur Berechnung der Ausgangswerte eines Terminals nur die Werte relevanter Eingänge heranzuziehen. Ein Eingang ist für den Wert eines Terminals genau dann relevant, wenn eine Änderung des Eingangs, unter Berücksichtigung der Verknüpfungslogik, eine Änderung des Ausgangs bewirkt. Diese Einschränkung der Wirksamkeit nicht relevanter Eingänge ist bei Entwürfen von Bedeutung, bei denen ausgenutzt wird, daß ein Terminal bei Änderung nicht relevanter Eingänge seinen Ausgangswert nicht ändert und deshalb weiterhin stabil bleibt. Diese Semantik der Terminal-Deklaration führt zu Verknüpfungstabellen für die logischen Funktionen, die an Stellen bei denen ein irrelevanter Eingang undefiniert ist, definierte Werte aufweisen. Für die Berechnung der Verzögerungszeit bedeutet dies, daß eine Änderung am Eingang eines Terminals nur dann Auswirkungen auf die Definiertheit des Ausgangs hat, wenn es sich um einen relevanten Eingang handelt. Die Verknüpfungstabellen für NAND und NOR sind im folgenden wiedergegeben, andere Verknüpfungen werden analog berechnet.

nand	0	s0	1	sl	u
0	1	1	1	1	1
s0	1	1	1	1	1
1	1	1	0	0	u
sl	1	1	0	0	u
u	1	1	u	u	u

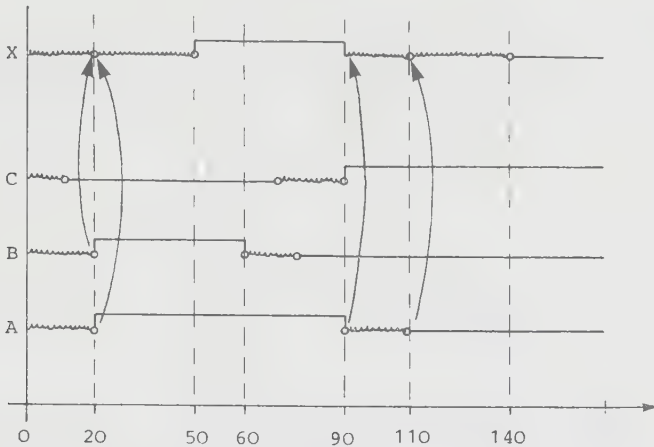
nor	0	s0	1	sl	u
0	1	1	0	0	u
s0	1	1	0	0	u
1	0	0	0	0	0
sl	0	0	0	0	0
u	u	u	0	0	u

Die Eigenschaften einer Terminal-Deklaration mit einer Verzögerungszeit von N Zeiteinheiten können in folgenden Stabilitätsbedingungen zusammengefaßt werden:

- o Der Ausgang eines Terminals ist genau dann definiert, wenn alle relevanten Eingänge seit mindestens N Zeiteinheiten stabil und definiert sind.
- o Werden relevante Eingänge eines Terminals undefiniert, wird der Ausgang ohne Verzögerung ebenfalls undefiniert.

Ein Beispiel für den zeitlichen Verlauf der Signale in einem Terminal ist in Bild 5.6 für folgende Terminal-Deklaration dargestellt:

```
terminal :30: X = (A nor B) nand C;
```



undefinierte Eingänge	Laufzeit des Terminals	Term. definiert	Änderung irrelevanter Eingänge	Änderung relevanter Eingang	Laufzeit des Terminals	definierter Ausgang
-----------------------	------------------------	-----------------	--------------------------------	-----------------------------	------------------------	---------------------

Bild 5.6: Zeitverhalten Terminal

Die Deklarationen von Terminals, die eine Selektion der Quellen abhängig von einer Selektionsvariablen enthalten, erfordern eine syntaktische Erweiterung der Zeitspezifikation. Da die verschiedenen Quellen eines Terminals im allgemeinen Verknüpfungen unterschiedlicher Komplexität enthalten, ist es notwendig, nicht nur der gesamten Deklaration eine

Verzögerungszeit zuzuweisen, sondern den einzelnen Quellen ebenfalls eine spezifische Verzögerungszeit zuzuordnen:

```
terminal :20: X =   if C then :30: A nor B
                   else :45: B nand (A nor C);
```

Im obigen Beispiel hat das Terminal X eine Verzögerungszeit von 50 (=20+30) Zeiteinheiten, wenn die Selektionsbedingung C erfüllt ist, sonst 65 (=20+45). Diese Zuordnung ermöglicht eine feinere Zeitspezifikation und vermeidet Meldungen des Simulators, die bei einer pauschaleren Spezifikation gelegentlich auftreten, wenn das Terminal auf einem kritischen Pfad liegt, der lediglich die schnellere Alternative verwendet.

Das in Abschnitt 4.1 dargestellte Syntaxdiagramm ist um die obigen Möglichkeiten der Zeitdeklaration entsprechend zu erweitern.

5.3. ZEITBESCHREIBUNG DER ZELLBIBLIOTHEK

Wie bereits in Abschnitt 1.3 ausgeführt, werden in der Graph-Grammatik der Zellbibliothek neben den Strukturinformationen und der Funktion der elementaren Zellen auch Zeitattribute spezifiziert. Diese Zeitattribute sind den Produktionen der Zellbibliothek zugeordnet und ermöglichen dem Synthesystem eine Berechnung der Verzögerungszeit für synthetisierte Verknüpfungen. Diese aufgrund der verwendeten Zellen berechneten Verzögerungszeiten müssen dann den, in der funktionalen Beschreibung vorgegebenen, Verzögerungszeiten entsprechen, wenn der in der funktionalen Beschreibung spezifizierte Schaltkreis mit Hilfe der Zellen aus der Zellbibliothek synthetisierbar sein soll.

5.3.1. Attribute für logische Verknüpfungen

Die in Bild 4.2 angegebenen Graphproduktionen und der Kontrollgraph in Bild 4.3 definieren den strukturellen Aufbau logischer Verknüpfungen und geben an, welche Kombinationen elementarer Zellen zu zulässigen Schaltungen führen. Die zeitlichen Eigenschaften der synthetisierten Schaltung werden durch zusätzliche, diesen Produktionen zugeordnete Attribute definiert. Die Verzögerungszeit eines Gatters ergibt sich dabei prin-

zipiell aus einer gatterspezifischen Verzögerung und der maximalen Verzögerung der Gattereingänge. Ergänzt man die Produktionen um die folgenden Attributfunktionen, kann die Gesamtverzögerung der logischen Verknüpfung durch Auswertung der Attribute bei der Synthese berechnet werden (vergleiche Bild 3.6 und 3.8).

Produktion:	Attributfunktionen:
Startgraph:	$\text{delay}(1, \text{IDENT}) \leftarrow \text{delay}(2);$
L1 (not):	$\text{delay}(1, \text{exp}) \leftarrow \text{delay}(2, \text{not}) + \text{delay}(1, \text{exp}),$ $\text{delay}(2, \text{not}) \leftarrow 1;$
L2 (nand):	$\text{delay}(1, \text{exp}) \leftarrow \text{delay}(2, \text{nand}) +$ $\quad \text{max}(\text{delay}(3, \text{exp}), \text{delay}(4, \text{exp})),$ $\text{delay}(2, \text{nand}) \leftarrow 3;$
L3 (nor):	$\text{delay}(1, \text{exp}) \leftarrow \text{delay}(2, \text{nor}) +$ $\quad \text{max}(\text{delay}(3, \text{exp}), \text{delay}(4, \text{exp})),$ $\text{delay}(2, \text{nor}) \leftarrow 1;$
L5 (and):	$\text{delay}(1, \text{exp}) \leftarrow \text{max}(\text{delay}(3, \text{exp}), \text{delay}(4, \text{exp}));$
L6 (or):	$\text{delay}(1, \text{exp}) \leftarrow \text{max}(\text{delay}(3, \text{exp}), \text{delay}(4, \text{exp}));$
L7 (input):	$\text{delay}(1, \text{exp}) \leftarrow \text{delay}(2, \text{IDENT}),$ $\text{delay}(2, \text{IDENT}) \leftarrow 0;$

Für die Beispiel 4.2 angegebene XNOR-Schaltung berechnet sich die Verzögerungszeit aufgrund der Attribute wie folgt:

$$\begin{aligned} \text{terminal :4: XNOR} &= (\text{not } A \text{ or not } B) \text{ nand } (A \text{ or } B); \\ 4 &= 3 + \text{max}(\text{max}(1+0, 1+0), \text{max}(0, 0)) \end{aligned}$$

Die alternative Definition in Beispiel 4.3 ergibt ein kompakteres Layout, da nur zwei statt drei Gatter benötigt werden, die Verzögerungs-

zeit ist jedoch größer:

```
terminal :6: XNOR =      (A or B) nand (A nand B);
          6 = 3 + max( max(0,0) , 3+max(0,0) )
```

Die Attribute der einzelnen logischen Funktionen hängen von der verwendeten Technologie ab. Für die NMOS-Technologie ist typisch, daß das NAND-Gatter eine größere Verzögerungszeit aufweist als das NOR-Gatter. Da die AND- und OR-Operationen nur innerhalb einer NAND- oder NOR-Verknüpfung auftreten können, verzögern diese die Schaltzeit nicht. Für die Berechnung ist lediglich das Maximum der Verzögerungszeiten der Eingänge relevant.

5.3.2. Attribute für PLA-Logik

Die Grundstruktur eines PLA, bestehend aus den Eingangsinvertern, den NOR-Gattern der AND- und OR-Ebene sowie den Ausgangsinvertern erlaubt eine relativ einfache Berechnung der Verzögerungszeit. Da die Ausgänge des PLA gemeinsame Terme besitzen, die wiederum zumindest teilweise die gleichen Eingangsvariablen aufweisen, sind die Verzögerungszeiten der einzelnen Ausgänge nahezu identisch. Es ist deshalb ausreichend, dem PLA insgesamt eine Verzögerungszeit zuzuordnen, die für alle Ausgänge gilt.

Die Schaltgeschwindigkeit des PLA wird durch die Anzahl der gemeinsamen Terme sowie der Ein- und Ausgänge bestimmt. Die in Bild 5.7 schematisch dargestellte PLA-Struktur zeigt den Einfluß der einzelnen Größen auf die Schaltgeschwindigkeit:

- (1) Die Anzahl der Terme in denen eine Variable auftritt, bestimmt den 'fan out' des Eingangsinverters und verlangsamt dadurch dessen Schaltgeschwindigkeit.
- (2) Die Anzahl der Variablen, die in einem Term auftreten, bestimmen die Transistoren des NOR-Gatters der AND Ebene ('fan in') und führen zu einer geringfügigen Beschleunigung.
- (3) Die Anzahl der Ausgänge, die durch die NOR-Gatter der AND Ebene angesteuert werden, bestimmen dessen 'fan out'. Mit wachsender

Anzahl der Ausgänge, in denen ein Term vorkommt, wird das PLA deshalb langsamer.

- (4) Der 'fan in' der NOR-Gatter der OR Ebene hängt von der Anzahl der Terme ab, die in einem Ausgang disjunktiv verknüpft sind. Mehrere Terme in einer Ausgangsgleichung führen zu einer geringen Beschleunigung.

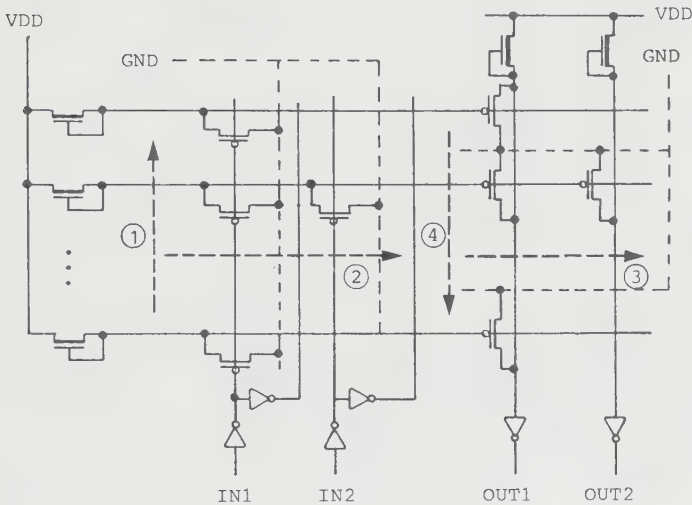


Bild 5.7: NOR-Gatter Struktur des PLA

Da der 'fan in' eines NOR-Gatters nur zu einer geringen Beschleunigung der Schaltgeschwindigkeit führt, werden die unter (2) und (4) beschriebenen Effekte bei der Berechnung der Schaltgeschwindigkeit vernachlässigt. Die Gesamtverzögerungszeit des PLA wird deshalb durch den 'fan out' der Eingangsinverter in der AND-Ebene (Anzahl der Terme, in denen ein Eingang auftritt) und den 'fan out' der NOR-Gatter der AND-Ebene (Anzahl der Terme, die zur Berechnung eines Ausgangs disjunktiv verknüpft werden) bestimmt. Die Schaltgeschwindigkeit eines einzelnen Gatters nimmt fast linear mit dem 'fan out' zu, in der hier zugrunde

gelegten NMOS-Technologie etwa mit 20%. Zur Berechnung der Schaltgeschwindigkeit werden die in Bild 4.18 angegebenen Produktionen um entsprechende Attribute ergänzt. Eine obere Grenze erhält man, wenn die Gesamtzahl der Terme und Ausgänge berücksichtigt werden. Dazu genügt es, die Produktionen PLA3 (zusätzliche Terme) und PLA2 (zusätzliche Ausgänge) mit globalen Attributen zu versehen, um die Gesamtzahl der Ausgänge und Terme zu zählen. Die Anzahl der Ausgänge ergibt sich aus der Häufigkeit, mit der die Produktion PLA2 (zusätzliche Ausgänge) angewendet wurde, die Anzahl der Terme erhöht sich mit jeder Anwendung der Produktion PLA3 (Terme, Mitte) um zwei, da aus Gründen der Zellgeometrie stets zwei neue Terme eingefügt werden.

```

Startgraph:    anzahl(term)    <-  2,
                anzahl(out)    <-  1;

PLA2:          anzahl(out)    <-  anzahl(out) + 1;

PLA3:          anzahl(term)    <-  anzahl(term) + 2;

```

Das in Bild 4.20 angegebene Beispiel der lokalen Steuerlogik enthält vier Terme (TERM = 4) und zwei Ausgänge (AUSGANG = 2). Legt man für die verwendete Technologie eine Inverterverzögerung von $t=5$ Zeiteinheiten zugrunde, erhält man die Gesamtverzögerung nach folgender Formel:

$$\begin{aligned}
 t(\text{PLA}) &= 2t + t * (\text{anzahl}(\text{term}) + \text{anzahl}(\text{out})) * \text{Faktor} \\
 &= 10 + 5 * (4 + 2) * 0,2 \\
 &= 16
 \end{aligned}$$

Die Verzögerungszeit des PLA beträgt damit 16 Zeiteinheiten und ist damit schneller als die in Kapitel 2 beschriebene Lösung mit NAND-Gattern. Da es sich um ein sehr kleines PLA handelt, ist der Flächenbedarf der Lasttransistoren und Inverter im Verhältnis zur Logik der AND- und OR-Ebene groß, was zu einer flächenmäßig ineffizienten Lösung geführt hätte. Für komplexe Verknüpfungen sind PLA's jedoch eine interessante Alternative, da neben einem kompakten Layout auch gute Schaltgeschwindigkeiten erzielt werden. Ein PLA mit 30 Termen und 10 Ausgängen hat nach obiger Formel eine Verzögerung von 50 Zeiteinheiten.

Eine genauere Berechnung der Verzögerungszeit des PLA würde nur die maximale Anzahl der Terme berücksichtigen, in denen ein Eingang vorkommt. Die Berechnung der Attribute erfordert dann jedoch eine kompliziertere Grammatik, da zur Berechnung nur jene Terme herangezogen würden, in denen die Eingangsvariable tatsächlich auftritt. Das gleiche gilt für die maximale Anzahl von Termen in den Disjunktionen der Ausgänge.

*The behaviour of a computer may
be described by an algorithm
like all sequential machines.*

F. Anceau

6. NICHTPROZEDURALE KONTROLLSTRUKTUREN FÜR DIE STEUERLOGIK

Betrachtet man digitale Schaltwerke auf der Gatterebene, handelt es sich um Netzwerke kontinuierlich aktiver Elemente, deren Trennung in Datenpfad und Logik zur Steuerung des Ablaufgeschehens nicht offensichtlich ist. Alle Komponenten des Entwurfs sind mit den gleichen Darstellungsmitteln beschrieben, die algorithmische Funktion ist nur schwer zu erkennen. Auf der Register-Transferebene wird, neben Speicher und Ein/Ausgabewerk zwischen dem Rechenwerk mit Registern sowie arithmetisch/logischer Einheit und dem Leitwerk zur Steuerung des Ablaufgeschehens unterschieden. Aktionen wie Datentransfers oder Verknüpfungen von Registerinhalten durch die arithmetisch logische Einheit werden zu bestimmten Zeitpunkten und in einer definierten Reihenfolge ausgeführt. Möglichst parallel, wenn dies konfliktfrei möglich ist, um eine hohe Leistung zu erzielen. Diese inhärente Parallelität verlangt ein adäquates Beschreibungsmittel sowohl für die Struktur, als auch die algorithmische Funktion des Entwurfs. Nichtprozedurale Entwurfssprachen entsprechen diesen Forderungen am besten.

Zur Beschreibung paralleler Abläufe werden in den folgenden Abschnitten zwei verschiedene, nichtprozedurale Kontrollstrukturen eingeführt, nämlich hierarchische Aktivierungsbedingungen und deterministische Automaten. Diese beiden Formen der Spezifikation des Kontrollflusses wurden gewählt, da beide typischen Realisierungstechniken der Steuerlogik hochintegrierter Schaltwerke entsprechen und somit bei der Modellierung technologischer Elemente auf der Register-Transferebene stets ein eindeutiger Zusammenhang mit den damit beschriebenen Implementierungstechniken gewahrt bleibt.

6.1. UHRGESTEUERTE TECHNIK

Im Gegensatz zu konventionellen Programmiersprachen, bei deren prozeduraler Beschreibungsweise die Sequenz der Ausführung von Anweisungen durch die Anordnung oder explizit mit Kontrollstrukturen bestimmt wird, ist bei nichtprozeduralen Sprachen die Ausführbarkeit einer Anweisung durch Aktivierungsbedingungen festgelegt ('guarded commands' DIJKSTRA 76).

Durch Verwendung der Uhr und aus ihr abgeleiteter Takte in den Aktivierungsbedingungen wird die Reihenfolge und die Parallelität der Ausführung von Anweisungen in CERES wie in anderen Rechner-Entwurfssprachen durch eine uhrgesteuerte Technik spezifiziert.

Diese Trennung von Notation und Ausführungsreihenfolge ermöglicht bei der Beschreibung eines Entwurfs eine strukturelle Anordnung von Deklarationen und Anweisungen, die der Nachbarschaft der Elemente im Layout der Schaltung entsprechen, ohne dadurch die Beschreibung des Ablaufgeschehens zu beeinflussen. Auf dem Gebiet der Rechnerentwurfssprachen haben deshalb nichtprozedurale Sprachen weite Verbreitung gefunden (DASGUPTA 82).

6.1.1. Parallele Abläufe

Die ältesten und bekanntesten Rechner-Entwurfssprachen CDL (CHU 65) und DDL (DIETMEYER 68) sind nichtprozedurale Sprachen, die zur Beschreibung des Zeitpunktes, bei dem ein Registertransfer stattfindet, Aktivierungsbedingungen verwenden. CASSANDRE (BOGO 71, MERMET 80) und ERES (KLAR 74, GARDILL 77) behalten dieses Konzept bei, sie erweitern jedoch die Spezifikation der Uhr, der Register und die der Terminals um eine realistische Zeitbeschreibung.

Grundgedanke der funktionalen Beschreibung ist die Spezifikation des Datenflusses zwischen Registern mit zwischengeschalteten logischen Verknüpfungen. Die Steuerung des Datenflusses erfolgt durch Aktivierungsbedingungen, die das Laden der Register überwachen (Ladetakt). Diese Aktivierungsbedingungen bestehen aus Takten, die mit weiteren logischen Signalen verknüpft sein können. Der Transfer neuer Werte in ein Register findet genau dann statt, wenn die Aktivierungsbedingung erfüllt ist. In

der Implementierung des Registertransfers entspricht sie dem Signal zur Aktivierung des Transfergatters am Registereingang des Zielregisters. Der gegenseitige Ausschluß von Registertransfers, die nicht gleichzeitig stattfinden sollen, muß durch geeignete Spezifikation der Aktivierungsbedingung gelöst werden.

Eine Anweisung der funktionalen Spezifikation besteht im einfachsten Fall aus Takten der Mehrphasenuhr als Aktivierungsbedingung, die in Schrägstrichen notiert wird ('/ .../'), gefolgt von einer Reihe von Mikrooperationen. Diese Mikrooperationen sind die elementaren Transferoperationen zwischen Registern, beziehungsweise zwischen einer logischen Verknüpfung und einem Register. Diese bereits in CDL eingeführte Grundstruktur wurde in verschiedenen Implementierungen geringfügig erweitert (LUNDSTRÖM 79). Die Lesbarkeit des Entwurfs wird durch eine Strukturierung der parallelen Aktionen verbessert, indem die, mit den Takten in einer Aktivierungsbedingung verknüpften, logischen Bedingungen als globale Kontrollstruktur formuliert werden. Dies entspricht der in (MOWLE 75) eingeführten Unterscheidung zwischen Haupt- und Nebenbedingung. Hauptbedingung ist der Takt, mit dem die Zielregister der Transferoperationen geladen werden, Nebenbedingungen (when .. do .. end) ergänzen diese und ermöglichen eine strukturierte Notation. Das folgende Beispiel beschreibt einen Teil des Steuerwerks für den Mikroprozessor OM2 (MEAD 80) in dieser erweiterten Notation.

"Deklarationen des OM2 (Ausschnitt):"

```

clock :0/50,150/300: PHI(1..2);

latch /LOAD,REFRESH/ ALU_IN_A = not not (( if LOAD then source)
                                wired ( if REFRESH then ALU_IN_A));

latch /LOAD,REFRESH/ ALU_OUT_A = not not (( if LOAD then source)
                                wired ( if REFRESH then ALU_OUT_A));

```

"Anweisungen des OM2 (Ausschnitt):"

```

/PHI(1), PHI(2)/          "PHI(1) ist Ladetakt"
begin      * * * *
  when SELECT_2 do
    ALU_IN_A <- case REG_SELECT_2 is
      8: A_BUS,
      9: SHARAY(16-SHIFT_CONST..31-SHIFT>CONST),
      10: SHIFT_CONST
    end
  end,
  when SELECT_1 do
    A_BUS <- .... ;
  end
end

/PHI(2), PHI(1)/          "PHI(2) ist Ladetakt"
begin
  ALU_OUT_A <- case RESULT_CONTROL is
    0: not PROPAGATE nand not CARRY,
    1: not PROPAGATE nand CARRY,
    2: PROPAGATE nand not CARRY,
    3: PROPAGATE nand CARRY
  end,

  when LATCH_FLAGS do
    FLAGS(5) <- FLAGS(15),
    .....
    FLAGS(10) <- ALU_OUT_A(0),
    FLAGS(12) <- ALU_OUT_A(15)
  end
end;

```

Beispiel 6.1: Funktionale Spezifikation des OM2 in CERES

Die Ablaufsteuerung des OM2 beruht auf einer zweiphasigen Uhr, die das Laden der Latch-Register für den Eingang und den Ausgang der arithmetisch logischen Einheit steuert. In Phase PHI(1) werden die Eingangsregister geladen, in Phase PHI(2) die Ergebnisse der Verknüpfung und die Ergebnisanzeigen übernommen. Da es sich in beiden Fällen um Latch-Register handelt, erfolgt die Wiederauffrischung jeweils mit der anderen Phase der Uhr. Abhängig von Selektionsbedingungen wird in jeder der Phasen eine Reihe von Transfers parallel ausgeführt. Diese Selektion wird durch zusätzliche Aktivierungsbedingungen spezifiziert. Die Syntax der entsprechenden CERES Anweisungen ist in Bild 6.1 dargestellt.

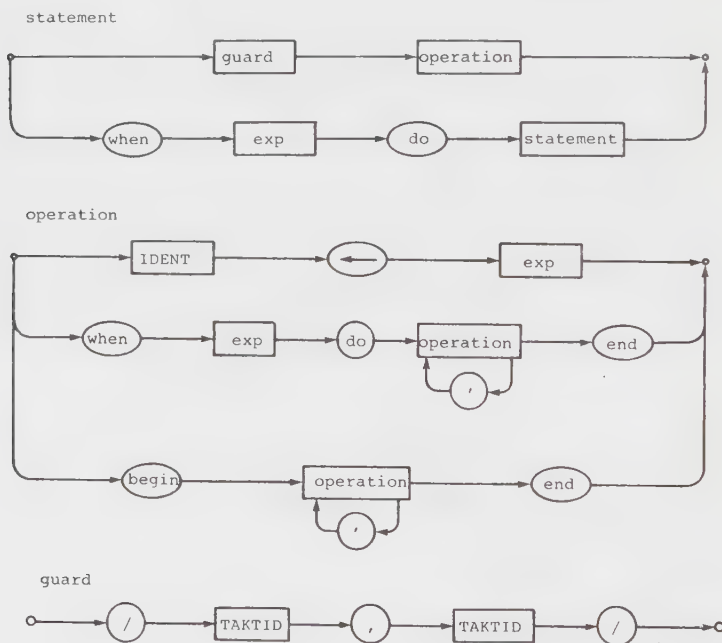
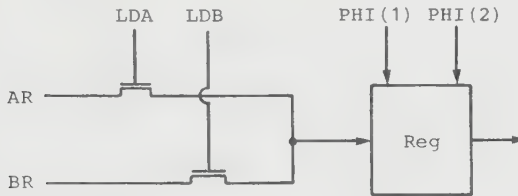


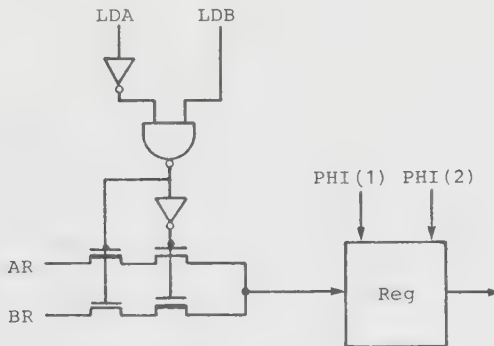
Bild 6.1: Syntaxdiagramm Anweisungen

Die Realisierung eines Register-Transfers, dessen Zeitpunkt durch eine

Aktivierungsbedingung spezifiziert ist, erfolgt durch Verbinden der entsprechenden Takteingänge des Registers mit den angegebenen Takten und der rechten Seite des Transfersoperators mit dem Registereingang (source). Um einen Konflikt bei der Beschaltung des Registers zu vermeiden, kann ein Register nur mit jeweils einem Takt für Lade- und Wiederauffrischungsphase beschaltet werden. Soll ein Register alternativ aus mehreren Quellen geladen werden, ist eine Selektionsbedingung zu spezifizieren. Dies kann durch implizites oder explizites Multiplexen geschehen.



a) Dual-Port Latch



b) Selektorschaltung am Latch-Eingang

Bild 6.2: Realisierung implizites und explizites Multiplexen

Beim impliziten Multiplexen wird das Register in mehreren Registertrans-

feroperationen als linke Seite verwendet, wobei verschiedene Aktivierungsbedingungen einen Schreibkonflikt ausschließen:

```
/PHI(1), PHI(2)/ begin
    when LDA do  REG <- AR end,
    when LDB do  REG <- BR end
end;
```

Bei einer alternativen Formulierung mit explizitem Multiplexen tritt das Zielregister nur einmal auf der linken Seite eines Register-Transfers auf, die Selektion der rechten Seite wird durch einen entsprechenden Ausdruck auf der rechten Seite des Transferoperators spezifiziert:

```
/PHI(1), PHI(2)/  REG <-  if (LDA and not LDB)
                        then AR else BR;
```

Die Realisierung von explizitem und implizitem Multiplexen zeigt Bild 6.2.

6.1.2. Sequentialisierung

Die Sequentialisierung der einzelnen Operationen kann durch die Verwendung einer Mehrphasenuhr, durch Zustandsregister oder durch eine Kombination beider Techniken erfolgen. Werden zur Steuerung des Kontrollflusses nur Aktivierungsbedingungen und Taktspezifikationen benutzt, sprechen wir von einer impliziten Sequentialisierung, da der Kontrollfluß nicht explizit spezifiziert wurde, sondern lediglich durch entsprechende Wahl von Steuerbedingungen definiert ist. Eine explizite Sequentialisierung von Anweisungen kann durch die, im nächsten Abschnitt einzuführenden, Kontrollstrukturen vorgenommen werden, welche die Beschreibung von Automaten gestatten. Der vom Entwickler vorgesehenen Kontrollfluß wird dabei durch entsprechende syntaktische Anweisungen spezifiziert.

6.2. Deterministische Automaten

Deterministische Automaten sind nicht nur Grundlage für die abstrakte Spezifikation der Maschinenzyklen in Mikroprozessoren, sondern werden

direkt als Hardware implementiert, indem der aktuelle Zustand in einem Register gespeichert wird und die Zustandsüberföhrungsfunktion durch PLA's realisiert wird (GUYOT 79). Dies erm6glicht einen 6nderungsfreundlichen und effizienten Entwurf. Die Maschinenzyklen werden wie beim Mikroprozessor INTEL 8085 (MALLADI 79) vielfach durch ein Phasenmodell erg6nzt. Die Interpretation eines Befehls erfolgt damit in mehreren Maschinenzyklen, die wiederum aus mehreren Phasen zusammengesetzt sind. Der dynamische Ablauf der Maschinenzyklen und Befehlsphasen wird dann mit einem Automatenmodell beschrieben (INTEL 79).

6.2.1. Funktionale Spezifikation

Die bekannteste Rechner-Entwurfssprache, die deterministische Automaten als Beschreibungsmittel f6r sequentielle Abl6ufe enth6lt, ist DDL (DIETMEYER 77), sie erlaubt jedoch keine Spezifikation des Zeitverhaltens. Das Zeitmodell in DDL beschr6nkt sich auf einen globalen Takt, der zum Fortschalten der Automaten w6hrend der Zustandsüberg6nge dient. Eine Weiterentwicklung stellt DDL-P dar (CLEEMPUT 81), bei der erstmals Daten- und Steuerteil getrennt spezifiziert werden. DDL-P wird durch das Programm APLAS erg6nzt, welches die Synthese eines PLA's aus einem, in DDL-P beschriebenen, Automaten unterst6tzt. Die auch in DDL vorhandene M6glichkeit mehrere Automaten gegenseitig aufzurufen, erfordert jedoch bei der Realisierung einen kellerartig organisierten Zustandsspeicher, der durch PLA's nicht implementiert werden kann. APLAS unterst6tzt deshalb den Wechsel der Zust6nde zwischen Automaten und die hierarchische Schachtelung von Automaten nicht. Der in (BROWN 81) beschriebene 'state-machine synthesizer' ist ein Entwurfssystem f6r deterministische Automaten vom MEALY-Typ, welches die Automatenbeschreibung in logische Gleichungen transformiert, die dann entweder diskret oder als PLA realisiert werden k6nnen.

Ein System zur Minimisierung von deterministischen Automaten, welche als hochintegriertes PLA realisiert werden sollen, ist in (TEEL 82) vorgestellt worden. In dieser Arbeit wird, ausgehend von einer Beschreibung des deterministischen Automaten auf der Register-Transferebene, ein Satz von Gleichungen generiert, der zur Minimisierung dient. Wesentlich an dieser Arbeit ist die gemischte MOORE/MEALY-Schreibweise der Automaten, die es dem Entwickler erm6glicht, sowohl zustands- als auch 6bergangsabh6ngige Ausgaben zu spezifizieren. Diese Notation erlaubt eine 6ber-

sichtliche Darstellung der Zustandsfolgen und der Ausgaben abhängig von den Eingaben, ohne daß eine MOORE/MEALY-Transformation durchgeführt werden muß.

Die funktionale Spezifikation eines Algorithmus durch deterministische Automaten ist ein von Entwicklern häufig verwendetes Mittel um komplexe Abläufe übersichtlich darzustellen. Bei vielen Anwendungen werden Automaten bereits in der Aufgabenstellung zur Darstellung der Funktion verwendet. Man denke zum Beispiel an Protokolle zur Datenübermittlung. Bei der Realisierung ist es deshalb von großer Bedeutung, daß die Entwurfssprache bereits Elemente enthält, die eine einfache Formulierung in der Terminologie der Aufgabenstellung ermöglichen. Die funktionale Simulation der Spezifikation in der Darstellung deterministischer Automaten erleichtert die Überprüfung der korrekten Funktion wesentlich.

Die direkte Abbildbarkeit der syntaktischen Elemente der Entwurfssprache auf entsprechende technologische Strukturen ist eines der Grundprinzipien des in dieser Arbeit entworfenen Synthese-Systems. Wie am Beispiel der Ampelsteuerung aus (MEAD 80) in den folgenden Abschnitten dargestellt, können deterministische Automaten auf einfache Weise durch PLA's implementiert werden, wenn diese um ein Register erweitert werden.

Die Einbeziehung deterministischer Automaten in die Entwurfssprache des Synthesystems hat für die algorithmische Beschreibung die gleiche Bedeutung, wie der Replikationsoperator für die strukturelle.

Das Syntaxdiagramm für die Spezifikation deterministischer Automaten (Bild 6.3) erweitert den in Bild 6.1 eingeführten Begriff 'statement'. Da eine Aktivierung des Automaten durch die Taktsignale des Registers erreicht wird, sind keine gesonderten Aktivierungsbedingungen zu spezifizieren.

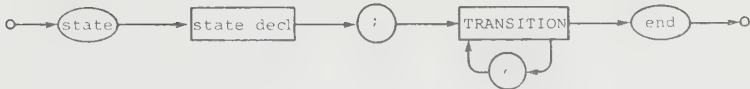
6.2.2. Beispiel Ampelsteuerung

Zur Erläuterung der Spezifikation deterministischer Automaten und ihrer Implementierung wird das Beispiel einer Ampelsteuerung verwendet, die den Verkehr an einer Kreuzung zwischen einer Landstraße und einer Nebenstraße regelt. Die Aufgabenstellung und ihre Lösung in NMOS-Technologie ist in (MEAD 79) ausführlich beschrieben, so daß diese hier nur skizziert wird.

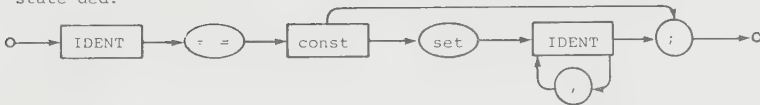
statement



state



state ded.



TRANSITION

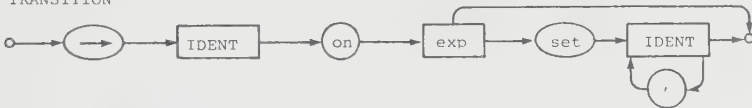


Bild 6.3: Syntaxdiagramm Automatenbeschreibung

Der Fernverkehr auf der Landstraße soll durch kreuzende oder einbiegende Fahrzeuge aus der Nebenstraße möglichst wenig gestört werden, so daß die Ampel der Nebenstraße nur geschaltet wird, wenn ein Fahrzeug durch Detektoren gemeldet wird ($C=1$). Dann wird zuerst die Ampel der Landstraße gelb und anschließend, zusammen mit der auf grün wechselnden Ampel der Nebenstraße, rot. Zu beachten ist bei diesem Beispiel, daß beim Wechsel von rot nach grün keine Gelbphase eingeschoben ist. Um den Verkehr auf der Landstraße nur kurzzeitig zu unterbrechen, wird die Ampel der Nebenstraße nach Ablauf einer kurzen Zeitspanne ($TS=1$) über gelb wieder nach rot geschaltet und kann frühestens nach Ablauf einer längeren Wartezeit ($TL=1$) wieder wechseln, auch wenn bereits vorher weitere Fahrzeuge auf der Nebenstraße ankommen. Bild 6.4 zeigt das Zustandsdiagramm des entsprechenden Automaten.

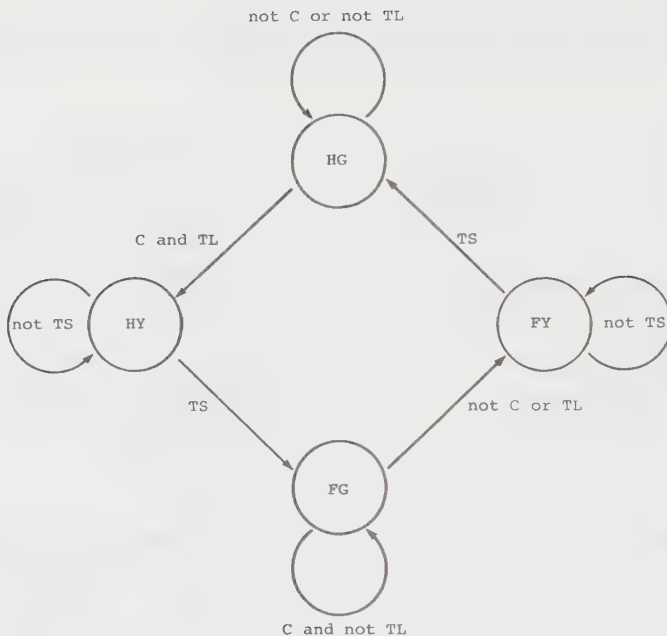


Bild 6.4: Zustandsdiagramm Ampelsteuerung (MEAD 80)

Die Implementierung eines deterministischen Automaten als PLA bedeutet die Konzentration der Steuerlogik in einem in sich abgeschlossenen Teil des Entwurfs. Da zwischen funktionaler Beschreibung und dem sich daraus ergebenden Layout neben der funktionalen auch eine strukturelle Übereinstimmung gegeben sein soll, sind innerhalb der Kontrollstruktur des Automaten keine anderen Anweisungen zulässig. Dies hat eine strenge Trennung zwischen Steuerlogik und dem gesteuerten Datenpfad zur Folge. Die Verbindung beider Komponenten erfolgt durch die Ausgänge des Automaten, die die entsprechenden Steuersignale für den Datenpfad repräsentieren.

Die Zustände des Automaten werden in einem Zustandsregister gespeichert, dessen Struktur und Taktsteuerung explizit deklariert werden muß. Der aktuelle Zustand ist ein Teil der Zustandsüberföhrungsfunktion, die

abhängig von Eingaben den Folgezustand und die entsprechenden Ausgaben bestimmt. Da die Codierung der Zustände die Größe des entstehenden PLA's beeinflusst, werden die einzelnen Zustände in der Spezifikation mit einer entsprechenden Codierung definiert. Am Beispiel der funktionalen Spezifikation der Ampelsteuerung sollen die einzelnen syntaktischen Elemente erläutert werden.

Die in (MEAD 80) vorgenommene Codierung der Ampelzustände im Register Y wurde beibehalten um einen besseren Vergleich zu ermöglichen. Die Farbe der Ampel von Landstraße (HL1,HLO) und Nebenstraße (FL1,FLO) sind in jeweils zwei Bit codiert, um eine Flächenreduktion des PLA zu erzielen. Dabei bedeutet das Setzen von HLO (bzw. FLO) die Ampel ist rot, das Setzen von HL1 (bzw. FL1) gelb. Ist keines der beiden Bits gesetzt, ist die Ampel grün.

Die zeitliche Steuerung der Ampelanlage erfolgt über eine Zeitschaltung, die gestartet wird (set ST) und dann nach kurzer Zeit (TS=1) sowie nach einer längeren Pause (TL=1) jeweils ein Zeitsignal zurückliefert.

```

controller
  register /PHI1, PHI2/ Y(0..1) = ...;      "Zustandsregister"

  state HG == 00  set FLO;                  "Highway green"
    -> HY  on (C and TL)  set ST,
    -> HG  on (not C or not TL)
  end,

  state HY == 01  set HL1, FLO;             "Highway yellow"
    -> FG  on TS  set ST,
    -> HY  on not TS
  end,

  state FG == 11  set HLO;                  "Farmroad green"
    -> FY  on (not C or TL)  set ST;
    -> FG  on (C and not TL)
  end,

  state FY == 10  set HLO, FL1;             "Farmroad yellow"
    -> HG  on TS  set ST,
    -> FY  on not TS
  end

end "Crossroad controller";

```

Die einzelnen Zustände des Automaten werden mit einem Bezeichner versehen, dessen Codierung im Zustandsregister bei der Spezifikation mit anzugeben ist. Eine mögliche Erweiterung wäre der Verzicht auf diese explizite Angabe der Zustandskodierung, die dann mittels eines Minimierungsverfahrens für die Anzahl der PLA-Terme automatisch zu bestimmen ist. Die Reihenfolge der Zustände ist bedeutungslos, mit Ausnahme des ersten Zustandes, der zugleich Anfangszustand ist.

Die zustandsabhängigen Ausgaben (MOORE-Automat) werden direkt bei der Zustandsdeklaration angegeben (`state .. set ..`), dabei nicht auftretende Ausgänge werden rückgesetzt. Die Folgezustände werden durch Rechtspfeile spezifiziert, die Übergangsbedingungen (`-> .. on ..`) dürfen nur Eingänge enthalten und müssen für alle Zustandsübergänge eindeutig spezifiziert sein. Damit sind nichtdeterministische Zustandsübergänge genau dann ausgeschlossen, wenn die Wertebereiche der Übergangsbedingungen disjunkt sind. Da das Verharren im gleichen Zustand ebenfalls explizit spezifiziert wird, muß jeweils genau eine Übergangsbedingung erfüllt sein. Zulässig ist dabei jedoch, daß ein Teil der Eingangsvariablen fehlt ('don't care' Signale).

Die übergangsabhängigen Ausgaben (MEALY-Automat) werden bei den entsprechenden Spezifikationen der Folgezustände notiert (`-> .. on .. set ..`). Diese gemischte Notation vereinfacht die Spezifikation der Automaten, da insbesondere beim Verharren im gleichen Zustand die bereits gesetzten, zustandsabhängigen Ausgaben erhalten bleiben.

6.2.3. Ermittlung der Zustandsüberföhrungsfunktion

Die funktionale Spezifikation eines Automaten kann in entsprechende Deklarationen und uhrgesteuerte Kontrollstrukturen transformiert werden. Im folgenden wird die Vorgehensweise am obigen Beispiel demonstriert, um das Prinzip der Transformation zu verdeutlichen.

Betrachtet man zunächst nur die Zustände des Automaten (codiert in $Y(0..1)$), sind für den Wert des Zustandsregisters nur jene Übergänge von Interesse, die zum Setzen wenigstens eines Bits im Zustandsregister führen. So wird $Y(0)$ in FG und FY, $Y(1)$ in FG und HY gesetzt. Die Bedingungen zum Setzen von $Y(0)$ ergeben sich aus dem aktuellen Zustand

und dem in der Übergangsbedingung angegebenen logischen Ausdruck. So wird aus den Übergängen:

```
state HG == 00 ... -> HY on (C and TL) ...
state HY == 01 ... -> FG on TS ...
                    -> HY on not TS ...
state FG == 11 ... -> FG on (C and not TL) ...
```

die Transfergleichung:

```
/PHI(1),PHI(2)/ Y(1) <- ( C and TL and not Y(0) and not Y(1) )
                        or ( TS and not Y(0) and Y(1) )
                        or ( not TS and not Y(0) and Y(1) )
                        or ( C and not TL and Y(0) and Y(1) );
```

Da Y(0) des Zustandsregisters bei den Zuständen FG und FY gesetzt ist, ergeben die Übergänge:

```
state HY == 01 ... -> FG on TS ...
state FG == 11 ... -> FY on (not C or TL) ...
                    -> FG on (C and not TL) ...
state FY == 10 ... -> FY on not TS ...
```

die Transfergleichung:

```
/PHI(1),PHI(2)/ Y(0) <- ( TS and not Y(0) and Y(1) )
                        or ( not C and Y(0) and Y(1) )
                        or ( TL and Y(0) and Y(1) )
                        or ( C and not TL and Y(0) and Y(1) )
                        or ( not TS and Y(0) and not Y(1) );
```

In ähnlicher Form werden die Gleichungen für die Ausgaben berechnet. Dabei sind die zustandsabhängigen Ausgaben nicht von den Übergangsbedingungen, sondern lediglich vom aktuellen Zustand und damit vom Wert des Zustandsregisters abhängig. So ergeben sich für die zustandsabhängigen Ampelsignale folgende Deklarationen:

```
terminal    FLO = (not Y(0) and not Y(1)) or (not Y(0) and Y(1)),
            FL1 = (Y(0) and not Y(1)),
            HLO = (Y(0) and Y(1)) or (Y(0) and not Y(1)),
```

$$HL1 = (\text{not } Y(0) \text{ and } Y(1));$$

Die Ausgaben, welche aufgrund eines Zustandswechsels erzeugt werden, enthalten neben der Übergangsbedingung auch den aktuellen Zustand des Automaten. Damit ergibt sich für ST folgende Deklaration:

```
terminal    ST = (C and TL and not Y(0) and not Y(1)
                  or (TS and not Y(0) and Y(1))
                  or (not C and Y(0) and Y(1))
                  or (TL and Y(0) and Y(1))
                  or (TS and Y(0) and not Y(1));
```

Werden obige Deklarationen und Anweisungen in der für ein PLA eingeführten Deklaration ('array logic') zusammengefaßt, indem man die Transfergleichungen für das Zustandsregister durch zwischengeschaltete Hilfs-terminals ersetzt, gelten die eingeführten Grammatiken zur Synthese des entsprechenden PLA's.

Stellt man diese Gleichungen tabellarisch dar, erhält man die (MEAD 80) angegebene Zustandsüberföhrungsfunktion. Dabei werden aus Optimierungsgründen gleiche Teilausdröcke in mehreren Gleichungen nur einmal aufgenommen und dafür mehrere Ausgänge gleichzeitig gesetzt. Andererseits kann aus diesen Gleichungen ohne weitere Umformungen mit Hilfe der in Abschnitt 4.3.3 angegebenen Graph-Produktionen direkt eine entsprechende PLA-Struktur synthetisiert werden. Das sich ergebende PLA ist ebenfalls in (MEAD 80) dargestellt.

Die aus der Transformation hervorgegangenen, komplexen Gleichungen zeigen, daß die Spezifikation einer Folge von Zuständen des zu entwerfenden Schaltkreises in Form von Automaten eine kompakte und übersichtliche Darstellung ermöglicht. Automaten sind ein bewährtes Mittel der Abstraktion auf der Register-Transferebene. Die effiziente Realisierung einer als deterministischer Automat spezifizierten Steuerung durch ein PLA und deren häufige Anwendung in hochintegrierten Schaltkreisen machen die Einführung der entsprechenden Kontrollstrukturen zu einem wesentlichen Bestandteil der Entwurfssprache.

Mehr, als wir schon vollbracht haben, ist noch übrig, aber ein großer Teil des Fortschritts ist es schon fortschreiten zu wollen.

SENECA

7. ZUSAMMENFASSUNG DER ERGEBNISSE UND AUSBLICK

Die Spezifikation eines hochintegrierten Schaltkreises wie des 'BUBBLE SORT' mit Hilfe einer Rechner-Entwurfssprache und die konsequente Ableitung sowohl der Topologie, als auch schaltungstechnischer Details aus der funktionalen Beschreibung, führen zu einem regulären Entwurf, dessen zu erwartende Leistung bereits während der Entwicklung bekannt ist.

Bisher war das größte Problem eines systematischen Entwurfs der Übergang zwischen verschiedenen Beschreibungsebenen, da die Darstellungsformen der einzelnen Ebenen kaum eine gemeinsame Basis erkennen liesen. Eine Synthese hochintegrierter Schaltkreise aus einer funktionalen Spezifikation erfordert jedoch ein auf allen Beschreibungsebenen gleichermaßen anwendbares Modell.

Der in dieser Arbeit entwickelte Ansatz zur Automatisierung dieses Entwicklungsprozesses durch ein Synthese-System verwendet als formales Hilfsmittel Graph-Grammatiken zur Beschreibung der syntaktischen und technologischen Strukturen und die Syntaxanalyse auf Graphen zur Formalisierung des Syntheseprozesses. Dies ermöglicht sowohl die Erfassung logischer und struktureller Eigenschaften des Entwurfs, als auch die Beschreibung zeitlicher Randbedingungen.

Um eine eindeutige Korrespondenz zwischen Entwurfs-Sprache und dem Syntheseprodukt herstellen zu können, die den Anforderungen hochintegrierter Schaltungen genügt, war es notwendig, Sprachelemente für häufig verwendete technologische Elemente wie quasistatische Register und Latches sowie Transfereatter einzuführen. Die Ergänzung der Kontrollstrukturen zur Beschreibung deterministischer Automaten erlaubt eine abstrakte Spezifikation der Steuerlogik auf der Register-Transferebene, deren Realisierung als PLA durch den Syntheseprozess einfach und effizient möglich ist. Durch Einführung des Replikationsoperators ist es

bereits in der funktionalen Spezifikation möglich, die Regularität des Syntheseprodukts zu beschreiben.

Die Validierung des Entwurfs auf der Register-Transferebene erfolgt durch Simulation der funktionalen Spezifikation. Da algorithmischer Ablauf und zeitliches Zusammenwirken der einzelnen Komponenten im Vordergrund der Betrachtung stehen, ist die Simulation umfangreicher und komplexer Entwürfe möglich, die in ihrer Gesamtheit auf technologieorientierten Beschreibungsebenen aus Aufwandsgründen nicht mehr untersucht werden können. Bei der beschriebenen Erweiterung der Entwurfssprache zur Modellierung hochintegrierter Schaltkreise ist es zweckmäßig, bereits auf der Register-Transferebene mit einer fünfwertigen Logik bei der Simulation zu arbeiten.

Die beim Entwurf des 'BUBBLE SORT' Bausteins gewählte Vorgehensweise, mit einer funktionalen Beschreibung auf der Register-Transferebene zu beginnen und den Entwurf durch Simulation zu validieren sowie seine Leistung zu verbessern, ist jedoch nicht nur in Hinblick auf die Entwicklung eines Synthese-Systems sinnvoll, sie ist vielmehr eine Entwurfstechnik, die bereits mit den vorhandenen Mitteln zu größerer Sicherheit bei der Entwicklung hochintegrierter Schaltkreise führt. Die bei Abschluß dieser Arbeit eingetroffenen ersten Muster des gefertigten 'BUBBLE SORT'-Chips konnten zwar wegen der noch nicht vorhandenen Test- und Messumgebung nicht auf ihre volle Funktionsfähigkeit untersucht werden, sie zeigen jedoch, daß eine systematische Entwicklung, beginnend auf einem hohen Abstraktionsniveau, mit relativ geringen Aufwand zu greifbaren Ergebnissen führen kann.

Im Rahmen dieser Arbeit wurde ein Konzept für die automatische Synthese hochintegrierter Schaltkreise aus einer funktionalen Beschreibung entwickelt, zu dessen Verwirklichung eine Reihe weiterer Arbeiten notwendig sind, um die verbliebenen Lücken und offenen Probleme zu klären. Die folgenden Abschnitte skizzieren Fragestellungen, die eine Einordnung dieser Arbeit ermöglichen und teilweise Gegenstand weiterer Forschungen sein müssen.

7.1. CERES, Rechner-Entwurfssprache für VLSI

Da es sich bei den vorgestellten Konstrukten der Rechnerentwurfssprache CERES um Erweiterungen von ERES handelt, sind diese in die entsprechende syntaktische Umgebung von ERES einzubetten. Auf die Darstellung eines vollständigen Syntaxdiagrammes der erweiterten Sprache CERES wurde verzichtet, um nicht den Eindruck der Vollständigkeit der diskutierten Entwurfs-Sprache zu erwecken. Ein einsatzfähiges Entwurfssystem kann auf die nicht behandelten Aspekte der Modularisierung und der Beschreibung von Speichern und anderer asynchroner Komponenten eines Entwurfes nicht verzichten. Die Modularisierung ist insbesondere im Hinblick auf reguläre Strukturen, bei denen gleiche Komponenten mehrfach verwendet werden, von besonderer Bedeutung. Ähnlich wie bei der Entwicklung umfangreicher Programme ist es auch bei hochintegrierten Schaltungen notwendig, auf bereits existierende Teilentwürfe mit bewährter Funktion und Leistung zurückzugreifen. Dies ist jedoch nur dann möglich, wenn vorhandene Module in den aktuellen Entwurf eingebunden und um problem-spezifische Teile ergänzt werden können.

Die in Kapitel 3 eingeführten quasihierarchischen Graphen beinhalten bereits die für eine Modularisierung notwendigen formalen Beschreibungsmittel. Ein aus verschiedenen Teilsystemen bestehender Entwurf kann als quasihierarchischer Graph aufgefaßt werden, bei dem die Verbindung der Schnittstellen durch entsprechende Kanten auf die internen Objekte der Moduln repräsentiert werden. Die Graph-Grammatiken sind dabei so zu erweitern, daß die verschiedenen Typen der Schnittstellensignale (in, out, bidirectional, asynchron..) bei der Komposition der Moduln zu einem umfassenderen Modul berücksichtigt werden.

Um einen Vergleich der ursprünglichen funktionalen Beschreibung des 'BUBBLE SORT' Bausteins in Kapitel 2 mit der entsprechenden, erweiterten Spezifikation in CERES zu ermöglichen, ist diese im Anhang wiedergegeben. Dabei wird deutlich, daß neben der detaillierten Deklaration der quasistatischen Register und Latches vor allem die Kontrollstrukturen zur Beschreibung der regulären Struktur und des funktionalen Ablaufs in der Steuerlogik neu eingeführt wurden. Die Grundzustände des Bausteins wie Laden, Sortieren und Entladen werden vollständig durch die externen Signale CHIPSEL und READ bestimmt. Wie bereits in Kapitel 2 ausgeführt, dient dies einer leichten Testbarkeit des Bausteins. Diese Zustände sind in den Terminals POPN und PUSHN codiert, die durch

entsprechende logische Gleichungen definiert wurden. Die für den Sortiervorgang charakteristischen, bedingten Register-Transfer-Operationen (when.. do ..) wurden in der ursprünglichen Version durch konjunktive Verknüpfungen der Taktsignale mit den entsprechenden Bedingungen spezifiziert, was keinen Hinweis auf die Realisierung durch Transfergatter ergab. Die Trennung von Takten und Aktivierungsbedingungen in CERES vermeidet semantische Probleme bei der Auswertung der Bedingungen, wenn diese zeitweise undefiniert sind.

7.2. Testfreundlicher Entwurf

Die Verwirklichung einer Entwurfsmethodik, die nicht nur bezüglich der Spezifikation korrekte Entwürfe liefert, sondern auch die leichte Testbarkeit der gefertigten Schaltung gewährleistet, erfordert die Beschränkung der Rechner-Entwurfssprache auf die Beschreibung und Simulation digitaler, synchroner Schaltwerke.

Grundprinzipien dieser Entwurfsmethodik sind Regeln zur Vermeidung dynamischer Probleme, die durch Wettlaufbedingungen von Signalen und Takten verursacht werden können. Diese Regeln wurden bei der Definition der Sprachelemente durch Graph-Produktionen berücksichtigt, ohne daß darauf näher eingegangen wurde. Sie werden deshalb hier zusammenfassend dargestellt und Vorschläge für Erweiterungen diskutiert.

Der Ausschluß rein kombinatorischer Rückkopplungen in synchronen Schaltwerken ermöglicht eine zeitdiskrete Simulation des Verhaltens, indem lediglich beim Wechsel von Taktsignalen eine Berechnung der Signalzustände erfolgen muß. Das Modell der Uhr und der Taktsignale erlaubt deshalb keine undefinierten Werte. Die Produktionen der Grammatik in Bild 4.2 erzeugen nur azyklische Graphen, die rückkopplungsfreien Schaltnetzen entsprechen. Zugleich werden dabei Wettlaufbedingungen, wie sie in asynchronen Schaltungen auftreten, vermieden und der Test der Schaltung mit einem automatischen Testsystem ermöglicht. Andernfalls müßte das Testsystem mit der Frequenz der Rückkopplung synchronisiert werden, was wegen der Streuung von Fertigungsparametern nicht möglich ist.

Die Spezifikation der Register und Latches sieht sowohl einen Lade- als auch einen Auffrischungstakt vor. Da lediglich die Registerdeklaration

eine, durch ein Transfergatter mit dem Auffrischungstakt getrennte, Rückkopplung gestattet, ist es unmöglich, andere als getaktete, speichernde Elemente zu verwenden. Dadurch wird eine in weiten Bereichen der Taktfrequenz konstante Funktion der gesamten Schaltung gewährleistet. Da die Testumgebung im allgemeinen nicht mit der vollen Frequenz der entworfenen Schaltung arbeiten kann, ist dies für die Durchführung einer automatischen Prüfung unumgänglich.

Die Ableitung von Taktsignalen (tick) aus Uhrtakten (clock) wurde restriktiv gehandhabt, um störungsfreie Taktsignale zu gewährleisten. Es ist lediglich das Mischen von Takten (nor-Verknüpfung) und das Sperren von Takten durch Datensignale (nand-Verknüpfung) sowie die Negation eines Taktsignales zulässig. Da bei hochintegrierten Schaltungen die parasitären Kapazitäten und Widerstände der Leitungen zu einer Verzögerung des Taktsignales führen können, die von der aktuellen Platzierung und Verdrahtung abhängen, ist eine konjunktive Verknüpfung zweier Taktsignale unzulässig. Das technisch realisierbare Multiplexen von Takten wurde ausgeschlossen, da die Funktion der Schaltung durch dynamische Bedingungen so grundlegend modifiziert werden kann, daß die Lesbarkeit der Spezifikation darunter leidet.

Die strenge Trennung von Takt und Datensignalen durch die Spezifikation der Aktivierungsbedingungen für die Register-Transfers gestattet weder die Verwendung von Datensignalen als Takt, noch das Anlegen von Takten an Dateneingängen. Die Mischung von Takt und Datensignalen würde eine Trennung in fünfwertige Logik für Datensignale und binäre Logik für Taktsignale in der Simulation unmöglich machen. Schwerwiegender ist jedoch, daß andernfalls die obigen Prinzipien zur Gewährleistung störungsfreier Taktsignale und die Beschränkung auf synchrone Schaltungen umgangen werden könnten.

Diese Regeln sind bei einer Erweiterung von CERES bezüglich modularer Systemstrukturen und asynchroner Kopplung von in sich synchronen Teilsystemen entsprechend zu ergänzen. Insbesondere bedarf die Verwendung von Tristate Ein-/Ausgängen und die Einbettung von Synchronisationsoperatoren für asynchrone Signale weiterer Untersuchung. Die Möglichkeit der Spezifikation einer Schaltung, die nichtgetaktete speichernde Elemente enthält (FLIP-FLOP, RAM, ROM), ist dabei besonders wichtig.

Jede Entwurfsmethodik bedeutet eine Beschränkung in der Verwendung der

durch die Technologie gebotenen Möglichkeiten. Wird dadurch eine höhere Fertigungsausbeute und Robustheit gegen Parameterstreuungen sowie eine höhere Produktzuverlässigkeit durch gute Testbarkeit erreicht, sind diese Einschränkungen gerechtfertigt.

7.3. Entwurf, Simulation und Messungen

Die in Bild 1.1 dargestellte Vorgehensweise beim Entwurf einer hochintegrierten Schaltung geht von einer in sich abgeschlossenen, vollständigen funktionalen Spezifikation des Bausteins aus. Für die Simulation der Funktion und des zeitlichen Verhaltens des Entwurfs ist es jedoch notwendig, die Systemumgebung, in die der Entwurf eingebettet werden soll, entsprechend einzubeziehen. Dies geschieht im allgemeinen durch Simulationsparameter, welche die Aktionen des Simulators steuern und Stimuli, welche die Schnittstelle des Entwurfs zu seiner Umgebung ersetzen.

Die Beschreibung des Verhaltens der Systemumgebung des zu simulierenden Entwurfs durch Stimuli ist nur bei sehr einfachen Schnittstellen zwischen Entwurf und Umgebung möglich. Im allgemeinen wird es notwendig sein, die Funktion der Systemumgebung zumindest in einem vereinfachten Modell ebenfalls zu beschreiben und in die Simulation mit einzubeziehen. Die Verwendung bereits existierender Moduln zur Beschreibung der Systemumgebung beeinflusst somit das Simulationssystem, da dieses für die Zuordnung der Stimuli dann nicht nur zwischen den einzelnen Signalen, sondern auch zwischen den Moduln unterscheiden muß.

Schwieriger gestaltet sich die Simulation, wenn die Funktion der Umgebung nur sehr ungenau bekannt ist und erst durch Messungen am existierenden Objekt ermittelt werden kann. Die Beschreibung von Funktion und zeitlichen Verhalten der Systemumgebung aufgrund der Messergebnisse kann dann aufwendiger werden als der eigentliche Entwurf. Eine wesentliche Vereinfachung läßt sich erzielen, wenn es gelingt, die gemessenen Daten direkt als Stimuli für die Simulation des Entwurfs einzusetzen. Dies bedeutet in letzter Konsequenz jedoch die Verbindung der Simulation des Entwurfs mit einer Stimulation und Messung der Systemumgebung durch eine entsprechende Hardware/Software-Schnittstelle. Diese Schnittstelle zwischen Simulation und Hardware-Messung wird bei der zunehmenden Ersetzung von Teilsystemen durch hochintegrierte Bau-

steine immer wichtiger werden, da dann bereits in der Entwurfsphase Aussagen über die Beeinflussung der Leistung des Gesamtsystems bei verschiedenen Alternativen des Entwurfs gemacht werden können. Diese Kombination von Messung und Modellierung würde eine neue Qualität der Bewertung zukünftiger Entwicklungen bedeuten.

7.4. Synthese, Platzierung und Verdrahtung

Der bei der Synthese des Entwurfs entstehende Ableitungsgraph enthält neben den Verbindungen der einzelnen Zellen auch Strukturinformationen, welche die hierarchische Komposition des Entwurfs repräsentieren. Im Rahmen dieser Arbeit konnte nicht untersucht werden, wie diese Information für eine automatische Platzierung und Verdrahtung der Zellen ausgenutzt werden kann. Konventionelle Algorithmen zur Platzierung gehen von einem Kräftemodell aus, das Zellen mit vielen gemeinsamen Verbindungen möglichst nahe beieinander platziert. Bei hierarchischen, regulären Entwürfen könnte die Ausnutzung der im Ableitungsgraphen vorhandenen Strukturinformation zu einer besseren Platzierung und damit zu kürzeren Verbindungen zwischen den Zellen führen. Die Untersuchung hierarchischer Platzierungsalgorithmen, die von der gesamten im Ableitungsgraphen vorhandenen Information Gebrauch machen, ist zur Ergänzung des Synthesesystems notwendig.

*Süß ist die Erinnerung
an vergangene Mühen.*

EURIPIDES

SCHLUSSBEMERKUNG

Die Überlassung der Programme des Systems LUCIE und die Möglichkeit zur Beteiligung am CMP 83 durch Prof. F. Anceau und A. Guyot haben die Voraussetzungen für den Entwurf und die Fertigung des 'BUBBLE SORT' Bausteins geschaffen. Die ständige Diskussionsbereitschaft und die vielen Anregungen aus dem Kreis meiner Kollegen sowie die kreative Atmosphäre am Lehrstuhl von Prof. Dr. U. Herzog haben wesentlich zum Entstehen dieser Arbeit beigetragen.

Für die Betreuung und konstruktive Kritik möchte ich mich an dieser Stelle bei Prof.Dr. U. Herzog und Dr. R. Klar, für die Übernahme des Zweitgutachtens bei Prof.Dr. H.J. Schneider bedanken.

Die vielfältige Unterstützung und Hilfsbereitschaft meiner Kollegen K.-H. John, U. Hehn, W. Dulz, Dr. W. Eberlein und Frau W. Schmielau, die Durchsicht des Entwurfs durch Frau Dr. M. Messerer und die von Frau C.Moog mit viel Geduld gezeichneten Abbildungen haben zum Gelingen dieser Arbeit beigetragen.

Meiner Frau Petra und meinen Söhnen Alexander und Dominik möchte ich für die Geduld und das Verständnis danken, das sie in den Monaten der Entstehung dieser Arbeit aufgebracht haben und mit der sie Arbeitsplatz-rechner, Drucker, Disketten, Plotterzeichnungen, Bücher, Ordner und Pfeifenqualm in unserem Wohnzimmer toleriert haben.

L I T E R A T U R V E R Z E I C H N I S

1. ENTWURFS- UND SYNTHESE SYSTEME

- (ANCEAU 81) Anceau F.: Parametres Electriques pour le Projet CMP
ENSIMAG Grenoble 1981
- (ANDERSON 62) Anderson J.P., Gorman D.F.: 'A logic design translator',
Proc. FJCC ,251-261; 1962
- (AYRES 79) Ayres R.: 'Silicon Compilation - a Hierarchical use of PLAs',
Proc. 16th Design Automation Conference 1979.
- (BARTSCH 83) Bartsch-Spörl B.: Grammatical Inference of Graph Grammars
for syntactic Pattern Recognition', in Ehrig H., Nagl M.,
Rozenberg G. (ed): 'Proc 2nd Int.WS on Graph Grammars and
their Application to Computer Science', Lect.Notes in Comp.Sc.,
Springer Berlin 1983
- (BIRZELE 82) Birzele P.: Bausteinentwurf auf der Basis von Zellen',
in Schwärtzel H.G. (ed): CAD für VLSI
Springer, Berlin-Heidelberg-New York 1982
- (BLAAUW 72) Blaauw G.A.: Computer Architektur',
Elektronische Rechenanlagen, Band 14, Nr.4 1972
- (BRENDL 79) Brendel W.: Formale Beschreibung von Programmiersprachen
durch Graph-Grammatiken,
Informatik FB 10, 57-74, Springer Verlag, Berlin 1979
- (BRENDL 82) Brendel W., Fan Z.G., Klar R., Schmielau W.: 'ERES 82, Handbook
and Case Study', Arbeitsber. IMMD vol.15/12, Erlangen 1982
- (BUNKE 83) Bunke H.: Graph Grammars as a Generative Tool in Image
Processing, in Ehrig H., Nagl M., Rozenberg G. (ed):
Proc 2nd Int.WS on Graph Grammars and their Application to
Computer Science, Lect.Notes in Comp.Sc., Springer Berlin 1983
- (BURSKY 82) Bursky D.: Software package lays out gate arrays completely',
Electronic Design, December 9, 1982
- (CLEEMPUT 83) vanCleemput W.M.: Bringing Semicustom IC Design In-House',
COMPUTER DESIGN, Feb. 1983
- (CLEEMPUT 76) vanCleemput W.M.: Computer Aided Design of Digital Systems,
a Bibliography', Vol. I-IV, Comp.Sc.Press, Woodland Hills 1976
- (DEMAN 82) DeMan H.J.: Mixed-Mode Simulation for MOS-VLSI: Why, Where
and How' Proc. ISCS 82, IEEE New York 1982

- (FILLER 84) Filler K.J.: 'Erweiterungen des CAD-Systems LUCIE um ein Programm zur geräteunabhängigen Ausgabe von Zeichnungen , Studienarbeit am IMMD 7 der Universität Erlangen 1984
- (GARDILL 77) Gardill R., Händler W., Heßling H., Klar R., Spiess P.P.: 'ERES, A Nonprocedural Computer Hardware Design Language with Precise Description of Timing', Arbeitsber. IMMD vol.10/15, Erlangen 1977
- (GÜTTLER 77) Göttler H.: 'Zweistufige Graphmanipulationssysteme für die Semantik von Programmiersprachen , Arbeitsbericht des IMMD Nr.10/12, Erlangen 1977
- (GRASS 79) Grass W., Lipp H.M.: LOGE - A highly effective System for Logic Design Automation , ACM SIGDA newsletter vol.9/2, 6-12, 1979
- (GRAY 79) Gray J.P.: 'Introduction to Silicon Compilation , Proc. 16th Design Autom.Conf. 219-220, 1979
- (GRAY 82) Gray J.P., Buchanan I., Robertson P.S.: Designing Gate Arrays using a Silicon Compiler , Proc. 19th Design Autom. Conf., 377-383, 1982
- (GUYOT 81) Guyot A., Jerraya A., Raymond J.: LUCIE, Language Universitaire Conception de circuits Integres pour l'Enseignement , ENSIMAG-Report, Grenoble 1981
- (GUYOT 83) Guyot A.: Le CMP 83', ENSIMAG, Equipe d Architecture des Ordinateurs, Grenoble 1984
- (HAFER 82) Hafer L.J., Parker A.C.: Automated Synthesis of Digital Hardware , IEEE Trans.on Comp. C 31/2, 93-109, 1982
- (JERRAYA 83) Jerraya A.: EXTRACT, Extracteur des Parametres Electriques ENSIMAG-Report, Grenoble 1983
- (JOHANNSEN 79) Johannsen D.: Bristle Blocks: A Silicon Compiler Proc. 16th Design Autom.Conf., 224-227, 1979
- (KELLER 82) Keller K.H., Newton A.R., Ellis S.: 'A Symbolic System for Integrated Circuits , Proc.19th Design Aut.Conf., NewYork 1982
- (KUNG 81) Kung H.T., Sproull B., Steele G.: VLSI Systems and Computations Springer, Berlin-Heidelberg-New York 1981.
- (LIPP 82) Lipp H.M.: 'Strukturiertes Entwerfen in der Digitaltechnik , NTZ Archiv, Bd.4 / 1, 1982
- (LOESCH 82) Loesch W.: Custom ICs from Standard Cells: A Design Approach , Computer Design, May 1982
- (LOSLEBEN 82) Losleben P.: Computer Aided Design for VLSI , in Barbe D.F.(ed): Very Large Scale Integration, Springer, Berlin-Heidelberg-New York 1982

- (LUHUKAY 82) Luhukay J., Kubitz W.J.: 'A Layout Synthesis System for NMOS Gate Cells', Proc 19th Design Autom. Conf., 307-314, 1982
- (MARWEDEL 79) Marwedel P.: 'The MIMOLA Design System: Detailed Description of the Software System', Proc. 16th Design Autom. Conf., 59-63, 1979
- (MEAD 80) Mead C.A., Conway L.: 'Introduction to VLSI-Systems', Addison-Wesley, Reading MA 1980
- (MOORE 79) Moore G.: 'VLSI: Some Fundamental Challenges', IEEE Spectrum, April 1979, pp.30-37
- (MOULLION 83) Moullion M.: 'Rechnergestütztes Entwerfen von CMOS-Gate-Arrays mit einer Verdrahtungsebene', Elektronik 7/April 1983
- (MUROGA 82) Muroga S.: 'VLSI System Design', Wiley & Sons, New York 1982
- (MUSGRAVE 82) Musgrave G.: 'Introduction to CAD applied to Electronics', IBM Summer Institute, Courchevel 1982
- (NEWTON 81) Newton A.R.: 'Computer-Aided Design of VLSI Circuits', Proc. IEEE, vol.69/10, Okt. 1981
- (PETERSON 81) Peterson J.G.: 'Keys to Successful VLSI System Design' in (KUNG 81)
- (RICE 79) Rice R.: 'Tutorial: VLSI, The Coming Revolution in Applications and Design', IEEE Spectrum, April 1979
- (ROTHERMEL 83) Rothermel H.-J., Mlynski D.A.: 'Verdrahtungskonzept für Makrozellen mit variabler Leiterbahnbreite', NTZ Archiv, Bd.4 / 1, 1982
- (RUPP 81) Rupp C.R.: 'Components of a Silicon Compiler System', in Gray J.P.: 'VLSI 81', Proc. 1st Int. Conf. on VLSI, Edinburgh 1981
- (SASTRY 82) Sastry S., Klein S.: 'PLATES: A Metric-Free VLSI Layout Language Conf. on Advanced Res. in VLSI, MIT 1982
- (SCHMIDT 80) Schmidt B.: 'GPSS-FORTRAN', John Wiley & Sons, New York 1980
- (SCHORR 64) Schorr H.: 'Computer Aided Digital Systems Design using a Register-Transfer Language', IEEE Trans. on Computers, EC 13/12, 730-737, 1964
- (SCHROEDER 84) Schröder S.: 'Entwurf und Implementierung eines graphischen Editors für VLSI-Layout', Studienarbeit am IMMD 7 der Universität Erlangen 1984
- (SCHWEIKERT 83) Schweikert D.G.: 'HIGHLAND - Design Aids for Gate Arrays and Augmented Standard Cell Designs', in Engl W.L. (ed):

- Großintegration', VDE-Verlag Berlin-Offenbach, NTG FB 82, 1983
- (SIBBERT 78) Sibbert H.: 'Dynamische, nichtlineare Schaltungssimulation', Höfflinger B.(ed): 'Großintegration', Oldenbourg, München 1978
- (SISKIND 82) Siskind J.M., Southard J.R., Crouch K.W.: 'Generating Custom high Performance VLSI Designs from succinct algorithmic Descriptions', Proc. MIT Conf. on advanced Research in VLSI, 28-39, 1982
- (SOUKUP 81) Soukup J.: 'Circuit Layout', Proc. IEEE, vol.69/10, Okt.1981
- (VERGNIERES 80) Vergnieres B.: 'Macro Generation Algorithms for LSI Custom Chip Design', IBM Journal 24, 5 Sept 1980
- (WERNER 82) Werner J.: 'The Silicon Compiler: Panacea, Wishful Thinking or old Hat', VLSI Design, 46-52, Sept/Oct 1982
- (WILLIAMS 83) Williams T.W., Parker K.P.: 'Design for Testability - A Survey', Proc. IEEE, vol.71/1, Jan. 1983
- (WIPFLER 83) Wipfler G.J., Mlynski D.A.: 'FOCUS - Ein kombinierter Kräfte und Schnitt-Platzierungsalgorithmus für den hierarchischen Entwurf integrierter Schaltungen', NTZ Archiv, Bd.4 / 1, 1982
- (ZIMMERMANN 79) Zimmermann G.: 'The MIMOLA Design System: A Computer Aided Digital Processor Design Method', Proc. 16th Design Autom. Conf., 53-58, 1979
- (ZINK 84) Zink K.: 'DESIREE: Automatische Prüfung von Entwurfsregeln für hochintegrierte Schaltkreise', Studienarbeit am IMMD 7 der Universität Erlangen 1984

2. VLSI - FALLSTUDIE SORT BAUSTEIN

- (ALEXANDER 77) Alexander D.R., Antione R.J., Brown G.W.: 'SPICE2 MOS Modelling Handbook', The BDM Corp., Albuquerque 1977
- (ANCEAU 81) Anceau F.: 'Parametres Electriques pour le Projet CMP ENSIMAG Grenoble 1981
- (ANCEAU 82) Anceau F., Reis R.: 'Complex integrated circuit Design Strategy', IEEE JSSC, SC 17/3, 458-464, 1982
- (BETH 83) Beth Th.: 'Codierung von Sprache', private Kommunikation 1983
- (BRANDT 83) Brandt K., Madas I., Mirbeth M., Zapf H.L.: 'Parameterbestimmung für MOS Modelle und rechnergestützte Dokumentation von Parametersätzen für Simulationszwecke', NTG-Fachtagung Großintegration Baden-Baden, NTG-Fachber. 82, VDE-Verlag 1983

- (CAREY 82) Carey M.J., Hansen P.M., Thompson C.D.: 'RESSI: A VLSI Implementation of a Record Sorting Stack', University of California, Report UCB/CSD 82/102, Berkeley 1982
- (GARDILL 77) Gardill R., Händler W., Heßling H., Klar R., Spiess P.P.: ERES, A Nonprocedural Computer Hardware Design Language with Precise Description of Timing', Arbeitsber. IMMD vol.10/15, Erlangen 1977
- (GUIBAS 82) Guibas L.J., Liang F.M.: Systolic Stacks, Queues and Counters', Conf. advanced Research in VLSI, MIT 82, 155-163
- (GUYOT 83) Guyot A.: 'Le CMP 83', ENSIMAG, Equipe d'Architecture des Ordinateurs, Grenoble 1984
- (MAVOR 83) Mavor J., Jack M.A., Denyer P.B.: Introduction to MOS LSI Design', Addison Wesley, London 1983
- (MEAD 80) Mead C.A., Conway L.: 'Introduction to VLSI Systems', Addison Wesley, London 1980
- (NEWTON 81) Newton A.R., Pederson D.D., Sangiovanni-Vincentelli A., Vladimirescu A., Zhang K.: 'SPICE Version 2G.5 User's Guide', University of California, Berkeley 1981
- (ULLMAN 80) Ullman J.: Principles of Database Systems', Computer Science Press 1980
- (WIRTH 76) Wirth N.: 'Algorithm + Data Structures = Programs', Prentice Hall, Englewood Cliffs 1976

3. ATTRIBUTIERTE GRAPH-GRAMMATIKEN

- (BARTSCH 83) Bartsch-Spörl B.: Grammatical Inference of Graph-Grammars for syntactic Pattern Recognition', in (EHRIG 83, S.1-7)
- (BRENDL 76) Brendel W.: Implementierung von Graph-Grammatiken, Arbeitsbericht IMMD Bd.9/1, 126-237; Erlangen 1976
- (BRENDL 77) Brendel W., Bunke H., Nagl M.: 'Syntaxgesteuerte Programmierung und inkrementelle Compilation', Informatik-Fachberichte 10, 57-74, Berlin: Springer 1977.
- (BRENDL 79) Brendel W.: Formale Beschreibung von Programmiersprachen durch Graph-Grammatiken', Informatik-Fachberichte 19, 218-230, Berlin: Springer 1979.
- (BRENDL 83) Brendel W.: Application of Attributed Graph-Grammars to VLSI Design, in Nagl M. (ed): Proc. WG 83, Workshop on Graphtheoretic Concepts in Comp. Science; Osnabrück, Juni 1983
- (BUNKE 83) Bunke H.: Graph Grammars as a generative tool in image

- understanding , in (EHRIG 83, S. 8-19)
- (EHRIG 73) Ehrig H., Pfender H., Schneider H.J.: Graph-Grammars: An algebraic approach', Proc. 14th Annual Conf.on Switching and Automata Theory 167-180, 1972
- (EHRIG 83) Ehrig H., Nagl M., Rozenberg G.: Graph-Grammars and their Application to Computer Science , Lecture Notes in Comp. Science 153; Berlin: Springer 1983.
- (GÜTTLER 77) Göttler H.: 'Zweistufige Graphmanipulationssysteme für die Semantik von Programmiersprachen , Arbeitsbericht IMMD 10/12; Erlangen 1977
- (GÜTTLER 78) Göttler H.: Semantical description by two-level graph-grammars for quasihierarchical graphs , Applied Comp. Science 13, 207-226, München: Hanser 1979.
- (GÜTTLER 83) Göttler H.: 'Attributed Graph Grammars for Graphics , in (EHRIG 83, S.130-142)
- (KAUL 83) Kaul M.: 'Parsing of graphs in linear time , in (EHRIG 83, S.206-218)
- (NAGL 74) Nagl M.: Formale Sprachen von markierten Graphen Arbeitsbericht IMMD, 7/4; Erlangen 1974
- (NAGL 79) Nagl M.: 'Graph-Grammatiken:Theorie,Anwendungen,Implementierung Vieweg; Braunschweig 1979
- (NAGL 83) Nagl M., Engels G., Gall R., Schäfer W.: Software specification by graph grammars', in (EHRIG 83, S.267-287)
- (PFALTZ 69) Pfaltz J.L.,Rosenfeld A.: Web grammars', Proc.Joint Conf.Art.Intelligence Washington 609-619, 1969
- (SCHNEIDER 70) Schneider H.J.: 'Chomsky-Systeme für partielle Ordnungen , Arbeitsbericht IMMD, 3/3; Erlangen 1970
- (SCHNEIDER 74) Schneider H.J.: Syntax-Directed Description of Incremental Compilers ,Lect.Notes Computer Science, 26, 192-201;Berlin 1974
- (SCHNEIDER 77) Schneider H.J.: Graph Grammars', Lect. Notes Computer Science, 56, 314-331; Berlin 1977
- (WILHELM 79) Wilhelm R.: Attributierte Grammatiken', Informatik Spectrum, Band 2, Nr. 3, S.123-130, Berlin 1979

4. SPEZIFIKATION UND SYNTHESE DES DATENPFADES

- (ANCEAU 82) Anceau F., Schoellkopf J.P.: CAPRI: A silicon Compiler for VLSI circuits specified by Algorithms , IBM Summer Institute,

Courchevel June 1982

- (AYRES 83) Ayres R.F.: 'VLSI: Silicon Compilation and the Art of Automatic Microchip Design', Prentice Hall London 1983.
- (BRENDL 82) Brendel W., Fan Z.G., Klar R., Schmielau W.: 'ERES 82 Handbuch und Fallstudie', Universität Erlangen, Arbeitsbericht des IMMD Band 15/12, September 1982.
- (BRENDL 83) Brendel W., Klar R.: 'Rechner-Entwurfssprachen', Universität Erlangen, Vorlesungsskriptum SS 1983
- (GARDILL 77) Gardill R., Händler W., Heßling H., Klar R., Spiess P.P.: 'ERES, A Nonprocedural Computer Hardware Design Language with Precise Description of Timing', Universität Erlangen, Arbeitsbericht des IMMD Band 10/15, November 1977.
- (GULDEN 83) Gulden S.: 'Erweiterung von ERES für hierarchische Modellierungen (System - Design - Subdesign)', Universität Erlangen, Diplomarbeit November 1983.
- (LEWIN 83) Lewin M.H.: 'Logic Design and Computer Organization', Addison Wesley, London 1983
- (MAVOR 83) Mavor J., Jack M.A., Denyer P.B.: 'Introduction to MOS LSI Design', Addison-Wesley London 1983.
- (MEAD 80) Mead C.A., Conway L.: 'Introduction to VLSI Systems' Addison-Wesley, Reading MA 1980.
- (SEGOVIA 82) Segovia P., Chuquillanqui S.: 'PAOLA: A Tool for Topological Optimization of large PLA', Proc. Design Automation Conf. 1982
- (WIRTH 76) Wirth N.: 'Algorithm + Data Structures = Programs', Prentice Hall, Englewood Cliffs 1976

5. DEFINITION UND SIMULATION DES ZEITMODELLS

- (ANCEAU 69) Anceau F., Liddel P., Mermet J., Payan C.: 'CASSANDRE: A Language to describe Digital Systems', Proc. 3rd Int.Symp. COINS, Miami 1969.
- (BRENDL 82) Brendel W., Fan Z., Klar R., Schmielau W.: 'Design Tuning with a CHDL. A case study for bitslice microprocessor simulation on the Registertransfer level', in Richter L., vanSpronsen C.J.(ed) 'MICROSYSTEMS: architecture integration and use', Proc. EUROMICRO 82, S.241-256, North Holland 1982
- (BRENDL 83) Brendel W., Deussing P., Klar R., Riehlein H.P.: 'Experiences with Designing a Microcomputer and its Programs by means of the Computer Hardware Description Language ERES', Int. Bericht 5/83, Univ. Erlangen IMMD 7, Erlangen 1983

- (KLAR 74) Klar R., Becker M., Spiess P.P.: 'The Erlangen Computer Design Language ERES (Erlanger Rechner Entwurfs Sprache)', Proc. Workshop on CHDL /176-185, 1974.
- (KLEINERT 83) Kleinert G., Lindner H.H.: Fertigungs- und Anwendungskriterien bei der Entwicklung von VLSI-Chips, NTG Fachbericht 82, VDE-Verlag Berlin 1983.
- (MURRMANN 79) Murrmann H.: 'Ausbeuteüberlegungen für hochintegrierte Halbleiterschaltungen', NTG Fachbericht 68, VDE-Verlag, Berlin 1979.
- (PILOTY 83) Piloty R., Barbacci M., Borriore D., Dietmeyer D., Hill F., Skelly P.: 'CONLAN Report', TH Darmstadt, Bericht RO 83/1, Darmstadt 1983.

6. KONTROLLSTRUKTUREN FÜR DIE STEUERLOGIK

- (BROWN 81) Brown D.W.: 'A State-Machine Synthesizer - SMS', Proc. 18th Design Automation Conf. 1981.
- (BOGO 71) Bogo H., Guyot A., Lux A., Mermet J., Payan C.: 'CASSANDRE: A language to describe Digital Systems', Proc. 3rd Int.Symp. COINS Miami 1969
- (CHU 65) Chu Y: 'An ALGOL-like Computer Design Language', Communications ACM 8,10 /607-615, 1965.
- (CLEEMPUT 81) van Cleemput W.M., Kang S.: Automatic PLA Synthesis from a DDL-P.Description, Proc. 18th Design Autom. Conf. 1981.
- (DASGUPTA 82) Dasgupta S: 'Computer Design and Description Languages', in Advances in Comp. Science, Academic Press 1982.
- (DIETMEYER 68) Dietmeyer D.L., Duley J.R.: 'A Digital Systems Design Language (DDL)', IEEE Trans. on Comp. 17,9 1968.
- (DIETMEYER 77) Dietmeyer D.L., Duley J.R.: 'Register Transfer Languages and their Translation', in Breuer (ed): Digital Systems Design Automation: Languages, Simulation and Data Base, Pitman, London 1977.
- (DIJKSTRA 76) Dijkstra E.W: A Discipline of Programming Prentice Hall, Englewood Cliffs 1976.
- (GARDILL 77) Gardill R., Händler W., Hessling H., Klar R., Spiess P.P.: ERES - Eine nichtprozedurale Rechner-Entwurfssprache mit präziser Zeitbeschreibung, Arbeitsbericht des IMMD, Univ.Erlangen Band 10,Nr.15, 1977
- (GUYOT 79) Guyot A.: Comparison of Z80 and INTEL 8085 Microprocessor,

IMAG Report 9/79, Grenoble 1979.

- (INTEL 79) INTEL Corp.: 'MCS-80/85 Family User's Manual',
INTEL Corporation 1979.
- (KLAR 74) Klar R., Becker M., Spiess P.P.: 'The Erlangen Computer Design
Language ERES (Erlanger Rechner-Entwurfs-Sprache)',
Proc. Workshop on CHDL S.176-185, 1974
- (LUNDSTRÖM 79) Lundström G.: 'A Survey over some Computer Hardware
Description Languages', Inst. of Techn., Uppsala Univ. 1979.
- (MALLADI 79) Malladi V.S.R.: 'Analysis of internal Architecture of
INTEL Microprocessors', IMAG Report 6/79, Grenoble 1979.
- (MEAD 80) Mead C., Conway L.: 'Introduction to VLSI Systems
Addison-Wesley 1980.
- (MERMET 80) Mermet J.: 'Functional Specification',
NATO Advanced Study Institut, Sogesta 1980
- (MOWLE 75) Mowle F.J., Stine L.R.: 'A Position Paper on the Extension to
the Computer Design Language', Proc. Int.Symp. on CHDL's 1975.
- (TEEL 82) Teel B., Wilde D.: 'A Logic Minimizer for VLSI PLA Design',
Proc. 19th Design Automation Conf. 1982.

STICHWORTVERZEICHNIS

Aktivierungsbedingung	163
Anweisungen	166
Architektur	5
BUBBLE SORT	55
- Algorithmus	55
- Grundriß	61
- Layout	76
- Spezifikation	57
CERES	43, 179
Deterministische Automaten	168, 171, 173
Entwurf	
- Methodik	9, 11, 29, 181
- Prozeß	5
- Regeln	32, 36
- Sprache	25
- testfreundlich	41, 180
- Zellen	12, 65
ERES	24
Gatter-Ebene	7
Gate Array	16
Graph-Grammatiken	46, 94, 113, 124, 137, 140, 141
- Ableitung	102, 107
- attributierte	97
- Implementierung	106
- Produktionen	98, 99, 100, 103
- programmierte	105
- Sprache	106
Kontrollgraph	104, 116, 142
Kundenschaltkreis	15
Latch	125, 127, 150, 152
Layout	29
- Ebene	8
- geometrisches	30
- Prüfung	35
- Sprache	30
- symbolisches	31

Logik	
- Attribute	156
- Elemente	154
- Felder	131
LUCIE	24
Operator	
- elementarer	111
- Replikationsoperator	131
PLA	131, 133
- Attribute	158
- Deklaration	134
- Schaltung	159
- Strichdiagramm	144
- symbolisches	135
- Zeitberechnung	160
- Zellen	138
Plazierung	183
Quasihierarchische Graphen	95, 97
Rechner-Entwurfssprachen	7, 25
Register	125
- Deklaration	128
- elektr. Simulation	81
- Layout	68
- Schaltung	66, 71, 80, 86, 126
- quasistatisch	126
- Zeitverhalten	150, 151
Register-Transferebene	7
Regularität	13, 130
Replikationsoperator	131
Selektion	167
Simulation	
- elektrisch	39, 77, 89
- funktional	27, 55, 62, 89, 91
- Zeitmodell	145
Silicon Compiler	43
Speicher	129
Spezifikation	
- BUBBLE SORT	57, 197
- funktional	10, 55, 169
- OM2	165

SPICE	24, 77, 78
Standard-Zellen	18
Steuerlogik	
- globale	44, 84
- Layout	73, 75
- lokale	69, 84, 135, 136, 143, 144
- Schaltung	70, 85
Syntaxanalyse	106
Synthese-System	43, 45, 183
Takt	
- abgeleiteter	145
- Deklaration	147
- Modellierung	149
- Spezifikation	147
Terminal	109
- Deklaration	110
- Selektion	120
- Verbindung	120
- Verzögerungszeit	153
- Zeitverhalten	155
Test	
- Methode	14
- Muster	40, 180
- Muster	40
Transistorebene	8
Transferrgatter	117, 118, 119
Uhr	
- Deklaration	149
- Mehrphasen	146
Vergleicher	60
- elektr.Simulation	83, 84
- Layout	68
- Schaltung	67, 82
VLSI	5
- Fallstudie	53
- Sortieralgorithmen	53
Zeitmodell	146
Zelle	
- allgemeine	20
- Bibliothek	22
- Entwurf	12, 65
- Konzept	14

ANHANG: CERES Beschreibung des BUBBLE-SORT Bausteins

```
design BUBBLE_SORT (in DATAIN(7..0), CHIPSEL, READ;
                   out DATAOUT(7..0), READY,
                   clock :0/240,320/100: PHI(1..2) );
```

"Globale Steuerlogik für die Zustände Laden, Sortieren und Entladen"

```
terminal :60:                                     "extern gesteuerte Funktionen"
    POPN    = CHIPSEL nand READ,
    PUSHN   = CHIPSEL nand not READ,
    READY   = not (PUSHN nand nor(SWAPRQ(,8)));
```

"Die Spezifikation des Datenpfades erfolgt nur für eine Ebene des Sortierfeldes, die anderen entstehen durch mehrfache Wiederholung für I=1..7 "

```
forall I= (1..7) is                               "Replikation der Ebenen"
  forall J= (8..1) is                             "Replikation der Bitslices"
    register :10: /PHI(1), PHI(2)/                "Sortier-Register"
      R(1..8,J) = not (if PHI(2) then not ((if PHI(1) then source)
                                         wired (if PHI(2) then R(1..8,J)))) );
```

```
terminal :20:                                     "Vergleicher-Element"
    SWAPRQ (I,J) = not ((R(I,J) and not R(I+1,J) nor
                       (SWAPRQ(I,J-1) and R(I,J)) nor
                       (SWAPRQ(I,J-1) and not R(I+1,J)) );
    SWAPRQ(I,0) = 0;                               "niederwertiges Carry"
endall J;
```

"Lokale Steuerlogik für eine Ebene des Sortierfeldes"

```
terminal :10: SWAP(I)  = SWAP(I-1) nand not SWAPRQ(I,8), SWAP(0) = 0,
              :20: UPIN(I)  = not PUSHN nand (POPN and (POPN nand SWAP(I))),
              :20: DOWNIN(I) = not POPN nand (PUSHN and (PUSHN nand SWAP(I)));
```

```
latch :10: /PHI(2),PHI(1)/                        "Steuer-Register für Datenpfad"
    UP(I) = not not ((if PHI(2) then source)
                    wired (if PHI(1) then UP(I) ),
    DOWN(I) = not not ((if PHI(2) then source)
                     wired (if PHI(1) then DOWN(I)));
```

"Steuerung des Datenpfades abhängig vom Zustand des Automaten"

```
/PHI(1), PHI(2)/ begin
    when not PUSHN do R(1,) <- DATAIN,
    when DOWN(I) do  R(I+1,) <- R(I,),
    when not POPN do DATAOUT <- R(1,),
    when UP(I) do    R(I,) <- R(I+1,)
end;

/PHI(2), PHI(1)/ begin
    UP(I) <- UPIN(I),
    DOWN(I) <- DOWNIN(I)
end;
```

```
endall I;
end "BUBBLE_SORT"
```


LEBENS LAUF

Wolfgang Brendel

- | | |
|-----------------|---|
| 4.11. 1951 | geboren in Fürth |
| 5. 9. 1958 | Eintritt in die Grundschule in Zirndorf |
| 9. 9. 1964 | Übertritt in die Peter-Vischer-Realschule
in Nürnberg |
| 7. 9. 1968 | Übertritt in das Hardenberg-Gymnasium Fürth |
| 19. 6. 1971 | Reifeprüfung am Hardenberg-Gymnasium Fürth |
| 1. 7. 1971 | Wehrdienst in Freising |
| 6.10. 1972 | Beginn des Studiums der Informatik an der
Technischen Fakultät der Friedrich-Alexander
Universität Erlangen-Nürnberg |
| 31.6. 1977 | Diplomprüfung in Informatik an der Technischen
Fakultät der Universität Erlangen-Nürnberg |
| 15.7. 1977 | Entwicklungs-Ingenieur bei der SIEMENS AG
in Erlangen |
| seit 1.10. 1981 | Wissenschaftlicher Mitarbeiter am Institut für
mathematische Maschinen und Datenverarbeitung
der Universität Erlangen-Nürnberg, Lehrstuhl
für Rechnerarchitektur und Verkehrstheorie |

In letzter Zeit sind folgende Arbeitsberichte des Instituts für
Mathematische Maschinen und Datenverarbeitung erschienen:

Band 13

=====

- Nr. 1 Schrammel, D., Bayer, W., Gliemann, H.-P., Heigl, M.,
Mertens, P.:
Ein Leistungs- und Wirtschaftlichkeitsvergleich zwischen
Klein- und Großcomputern
(Januar 1980)
- Nr. 2 Göttler, H.:
Semantische Modelle
(März 1980)
- Nr. 3 Fromm, H.J.:
Zur Modellierung der Speicherinterferenz bei hierarchisch
organisierten Multiprozessorsystemen
(April 1980)
- Nr. 4 Jäpel, D.:
Klassifikatorbezogene Merkmalsauswahl
(Juli 1980)
- Nr. 5 Bodendorf, F., Mertens, P., Haas, R., Weber, G., Wolf, G.:
SAMBA - ein Methodenbankrahmen um das Statistik-Paket SPSS
(Juli 1980)
- Nr. 6 Bartsch, B.:
Inferenz und Analyse spezieller Graphgrammatiken für die
syntaktische Mustererkennung
(September 1980)
- Nr. 7 Wurm, F.X.:
Auftragssystem für eine Mehrprozessoranlage
(Oktober 1980)
- Nr. 8 Schedelbeck, A.:
Vergleich der beiden höheren Realzeitprogrammiersprachen
ADA und PEARL
(Oktober 1980)
- Nr. 9 Jahresbericht 1980 der Informatik
(März 1981)

- Nr. 1 Pattern Recognition
 (Research at Lehrstuhl für Informatik 5)
 (Februar 1981)

- Nr. 2 Bunke, H.:
 Analyse elektrischer Schaltpläne
 (März 1981)

- Nr. 3 Linster, C.-U.:
 SYMPOS/UNIX - Ein Betriebssystem für homogene
 Polyprozessorsysteme
 (Juni 1981)

- Nr. 4 Akyildiz, J.F., Bolch, G.:
 Analytic Solution Techniques for Queueing Network
 Models of Computer Systems
 (Juli 1981)

- Nr. 5 Lindstedt, W.G.:
 Ein Verfahren zur Erstellung portabler algorithmischer
 Laufzeitprogramme
 (Juli 1981)

- Nr. 6 Vollmar, R.:
 Zum Begriff des Parallelismus bei Polyautomaten
 (Juli 1981)

- Nr. 7 Luft, A.L.:
 Rationaler Sprachgebrauch und orthosprachliche
 Standardisierung als Grundlagen zuverlässiger Software-
 Entwicklung und -Dokumentation
 (Januar 1982)

- Nr. 8 Händler, W.:
 PROCEEDINGS OF A WORKSHOP ON TAXONOMY IN COMPUTER
 ARCHITECTURE
 (Februar 1982)

- Nr. 9 Beth, Th., Hess, P., Wirl, K.:
 Materialien zur Kryptographie
 (März 1982)

- Nr. 10 Jahresbericht 1981 der Informatik
 (März 1982)

- Nr. 1 Wittmann, A.:
Ein Mechanismus für die Synchronisation paralleler
Prozesse
(Februar 1982)
- Nr. 2 Maehle, E.:
Fehlertolerantes Verhalten in Multiprozessoren -
Untersuchungen zur Diagnose und Rekonfiguration
(März 1982)
- Nr. 3 Bathelt, P.:
Vergleich von Synchronisationsmechanismen
(Mai 1982)
- Nr. 4 Keramidis, S.:
Eine Methode zur Spezifikation und korrekten
Implementierung von asynchronen Systemen
(Juni 1982)
- Nr. 5 Fromm, H.J.:
Multiprozessor-Rechenanlagen:
Programmstrukturen, Maschinenstrukturen und Zuordnungs-
probleme
(Mai 1982)
- Nr. 6 Bley, H.:
Vorverarbeitung und Segmentierung von Stromlaufplänen
unter Verwendung von Bildgraphen
(Juli 1982)
- Nr. 7 Hchl, W.:
Informatik-Sammlung
Katalog mit historischen Erläuterungen
(August 1982)
- Nr. 8 Messerer, M.:
Ein neuer Ansatz zur Parallelisierung von
Compilern
(August 1982)
- Nr. 9 Grosch, J.:
Eine Programmiersprache mit mengentheoretischen
Konstrukten und deren effiziente Implementierung
(August 1982)
- Nr. 10 Kleinöder, W.:
Stochastische Bewertung von Aufgabenstrukturen
für hierarchische Mehrrechnersysteme
(August 1982)

Band 15 Fortsetzung

=====

- Nr. 11 Kneißl, F.:
Realisierung von Makro-Datenflußmechanismen auf
hierarchischen Mehrrechnersystemen
(August 1982)
- Nr. 12 Brendel, W., Fan, Z., Klar, R., Schmielau, W.:
ERES 82
Handbuch und Fallstudie
Handbook and Case Study
(Oktober 1982)
- Nr. 13 Pelz, K.:
Portable Codegenerierung in Übersetzern
für Prozeß-Programmiersprachen
(Oktober 1982)
- Nr. 14 Sommer, U., Steinchen, J.:
Ein Kleinrechner-Grafiksystem zur Dokumentation
von EDV-Abläufen
(Oktober 1982)
- Nr. 15 Hein, H.-W.:
Das Erlanger Spracherkennungssystem
(November 1982)
- Nr. 16 Bolch, G., Bruchner, W.:
Analytische Modelle für symmetrische Mehrprozessor-
anlagen mit dynamischen Prioritäten
(Dezember 1982)
- Nr. 17 Jahresbericht 1982 der Informatik
(März 1983)

- Nr. 1 Gall, R.:
Formale Beschreibung des inkrementellen Programmierens-
im-Großen mit Graph-Grammatiken
(März 1983)
- Nr. 2 König, R.:
Beiträge zur algebraischen Theorie der formalen Sprachen
(März 1983)
- Nr. 3 Welbourn, D.B.:
The Design of Mechanical Components and the Development
of DUCT:
17 Years of CAD/CAM in Cambridge University
(April 1983)
- Nr. 4 Woitok, R.:
Die System-Implementierungs-Sprache ALICE
(Mai 1983)
- Nr. 5 Schneider, S., Schwab, P., Renninger, W.:
Wesen, Vergleich und Stand von Software zur Produktion
von Systemen der computergestützten Unternehmensplanung
(Juli 1983)
- Nr. 6 Händler, W., Herzog, U., Hofmann, F., Schneider, H.J.
und Mitarbeiter:
Projektabschlußbericht 1978 - 1982 Erlangen General Purpose
Array
(Juli 1983)
- Nr. 7 Mackert, L.:
Modellierung, Spezifikation und korrekte Realisierung
von asynchronen Systemen
(Juli 1983)
- Nr. 8 Reitenspieß, M.:
Sprachkonstrukte zur Spezifikation und korrekten Implemen-
tierung von Schutzproblemen
(Juli 1983)
- Nr. 9 Weber, K.:
Modellierung von Fehlverhalten mit Berücksichtigung
paralleler Abläufe
(August 1983)
- Nr. 10 Dietsch, H.:
Ein verteiltes Kommunikationssystem für einen Verbund
aus lokalen Mikrorechner-Netzwerken
(Dezember 1983)

- Nr. 11 Maehle, E., Schmitter, E.:
Workshop Fehlertolerante Mehrprozessor- und Mehr-
rechnersysteme
(Dezember 1983)
- Nr. 12 Volkert, J.:
Multiplikative Abstandsfunktionen und deren Anwendung
in der Bildverarbeitung
(Dezember 1983)
- Nr. 13 Steinbauer, D.:
Transaktionen als Grundlage zur Strukturierung und
Integritätssicherung in Datenbank-Anwendungssystemen
(Dezember 1983)
- Nr. 14 Wilke, P.:
Zusammenhänge und Unterschiede zwischen Graph-Grammatiken
und Petri-Netzen sowie verwandter Systeme
(Dezember 1983)
- Nr. 15 Jahresbericht der Informatik 1983
(März 1984)

- Nr. 1 Eberlein, W.:
Architektur technischer Datenbanken für
integrierte Ingenieursysteme
(Februar 1984)
- Nr. 2 Mertens, P., Heigl, M.:
Neuere Entwicklungen der computergestützten
Produktionsplanung, Eignung - Verbindungen -
Entwicklungspfade
(April 1984)
- Nr. 3 Stoyan, H.:
Maschinen- unabhängige Code-Erzeugung als semantik-
erhaltende beweisbare Programmtransformation
(Juli 1984)

